(19)日本国特許庁（ＪＰ）　(12)公表特許公報（Ａ）　(11)特許出願公表番号
特表2002－517038
（P2002－517038A）
(43)公表日　平成14年６月11日(2002.6.11)

(51)Int.Cl.⁷	識別記号	ＦＩ		テーマコード（参考）
Ｇ０６Ｆ　17/16		Ｇ０６Ｆ　17/16	Ｄ	５Ｂ０３３
			Ｓ	５Ｂ０５６
9/34	３３０	9/34	３３０	

審査請求　未請求　予備審査請求　有　（全 96 頁）

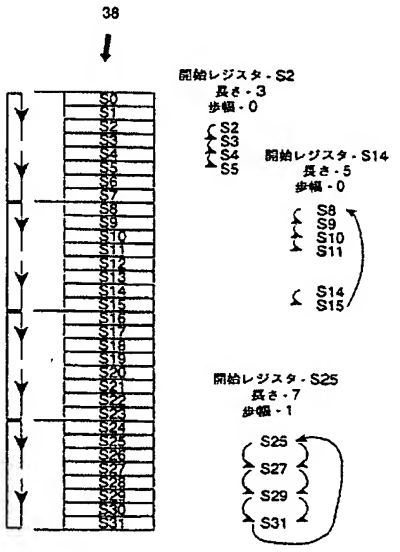
(21)出願番号　特願2000－551329(P2000－551329)
(86)(22)出願日　平成11年３月９日(1999.3.9)
(85)翻訳文提出日　平成12年11月24日(2000.11.24)
(86)国際出願番号　ＰＣＴ／ＧＢ９９／００７０７
(87)国際公開番号　ＷＯ９９／６１９９７
(87)国際公開日　平成11年12月２日(1999.12.2)
(31)優先権主張番号　０９／０８５，７５２
(32)優先日　平成10年５月27日(1998.5.27)
(33)優先権主張国　米国（ＵＳ）
(81)指定国　ＥＰ(ＡＴ，ＢＥ，ＣＨ，ＣＹ，ＤＥ，ＤＫ，ＥＳ，ＦＩ，ＦＲ，ＧＢ，ＧＲ，ＩＥ，ＩＴ，ＬＵ，ＭＣ，ＮＬ，ＰＴ，ＳＥ)，ＣＮ，ＩＬ，ＩＮ，ＪＰ，ＫＲ，ＲＵ

(71)出願人　エイアールエム　リミテッド
イギリス国　シーピー１　９エヌジェイ　ケンブリッジ，チェリー　ヒントン，フルバーン　ロード　110
(72)発明者　ヒンドス、クリストファー、ニール
アメリカ合衆国　テキサス、オースチン、パック　ベイ　レーン　6400
(72)発明者　ジャガー、デビッド、ビビアン
アメリカ合衆国　テキサス、オースチン、ガルスワーシイ　コート　5704
(74)代理人　弁理士　浅村　皓　（外３名）

最終頁に続く

(54)【発明の名称】　再循環レジスタファイル

(57)【要約】
複数のレジスタを含むレジスタバンクを有する浮動小数点ユニットは、異なるレジスタの一連のデータ値に対して指定された演算を複数回実行するベクトル演算をサポートする。レジスタバンクはサブセットに分割され、ベクトル演算に使用される一連のレジスタがサブセット内で回り込む。サブセットは連続したレジスタ番号の、互いに素な複数グループを有する。レジスタグループ内の回り込みによって、ＦＩＲフィルタリングおよびマトリックス変換等、ＤＳＰ演算を実行するためのコンパクトで効率的なコードを提供することができる。



【特許請求の範囲】

【請求項１】　データ処理装置であって、

複数のレジスタを有するレジスタバンクと、

前記レジスタバンク内の一連のレジスタのデータ値を使ってデータ処理演算を複数回実行するベクトル演算を指示する少なくとも１つのデータ処理命令に応動する命令デコーダとを備え、

前記レジスタバンクは少なくとも１つのレジスタサブセットを含み、前記一連のレジスタが前記サブセット内に存在し、

前記命令デコーダは前記一連のレジスタが前記レジスタサブセット内で回り込むように前記一連のレジスタを制御する、ことを特徴とするデータ処理装置。

【請求項２】　前記ベクトル演算は、複数連のレジスタの対応する複数のデータ値を使って前記データ処理演算を実行し、

前記レジスタバンクは複数のレジスタサブセットを含み、前記複数連のレジスタがそれぞれのサブセット内に存在し、

前記命令デコーダは前記複数連のレジスタがそれぞれのレジスタサブセット内で回り込むように前記複数連のレジスタを制御する、ことを特徴とする請求項１に記載の装置。

【請求項３】　前記複数のサブセットは互いに素である、ことを特徴とする請求項２に記載の装置。

【請求項４】　前記サブセットは、連続して番号付けされたレジスタからなる１つのレジスタグループを有する、ことを特徴とする請求項１、請求項２および請求項３のいずれかに記載の装置。

【請求項５】　前記複数のサブセットの各々は、連続して番号付けされたレジスタからなる１つのレジスタグループを有する、ことを特徴とする請求項２に記載の装置。

【請求項６】　前記複数のサブセットは、連続して番号付けされたレジスタからなる各レジスタグループが連続している複数のレジスタグループを有する、ことを特徴とする請求項５に記載の装置。

【請求項７】　前記複数のサブセットは、４つの連続するレジスタグループ

を有する、ことを特徴とする請求項６に記載の装置。

【請求項８】　メモリと、前記メモリと前記レジスタバンク内のレジスタとの間でのデータ値の転送を制御する転送コントローラと、をさらに有し、前記転送コントローラは複数の転送命令に応動して前記メモリと前記レジスタバンク内の一連のレジスタとの間で一連のデータ値を転送する、ことを特徴とする請求項１～請求項７のいずれかに記載の装置。

【請求項９】　各レジスタグループはそのレジスタグループの両端間で回り込みを行うインクリメンタを介してアドレスでアクセスされる、ことを特徴とする請求項６記載の装置。

【請求項１０】　前記一連のレジスタとは連続した一連のレジスタである、ことを特徴とする請求項１～請求項９のいずれかに記載の装置。

【請求項１１】　前記レジスタバンクおよび前記命令デコーダは浮動小数点ユニットの一部である、ことを特徴とする請求項１～請求項１０のいずれかに記載の装置。

【請求項１２】　レジスタバンクの複数のレジスタ内にデータ値を格納するステップと、

ベクトル演算を指示する少なくとも１つのデータ処理命令に応動して、前記レジスタバンク内の一連のレジスタのデータ値を使ってデータ処理演算を複数回実行するステップを有するデータ処理方法であって、

前記レジスタバンクは少なくとも１つのレジスタサブセットを含み、前記一連のレジスタが前記サブセット内に存在し、

前記実行中に、前記一連のレジスタは前記レジスタサブセット内で回り込む、ことを特徴とするデータ処理方法。

【請求項１３】　前記ベクトル演算は複数連のレジスタの対応する複数のデータ値を使って前記データ処理演算を実行し、

前記レジスタバンクは複数のレジスタサブセットを含み、前記複数連のレジスタがそれぞれのサブセット内に存在し、

前記実行中に、前記複数連のレジスタは前記レジスタサブセット内で回り込む、ことを特徴とする請求項１２に記載のデータ処理方法。

【請求項１４】　一連のレジスタ内のデータ値はフィルタのタップ係数であり、別の一連のレジスタ内のデータ値は前記フィルタによってフィルタされる信号値である、ことを特徴とする請求項１３に記載のデータ処理方法。

【請求項１５】　少なくとも一連のレジスタの開始点をベクトル演算毎に変えることによって、前記複数連のレジスタ内のデータ値に対して複数のベクトル演算を実行する、ことを特徴とする請求項１２に記載のデータ処理方法。

【発明の詳細な説明】

【０００１】

本発明はデータ処理の分野に関する。より詳しくは、本発明はレジスタバンクを有しかつベクトル演算をサポートするデータ処理システムに関する。

【０００２】

レジスタバンクを有しかつベクトル演算をサポートするデータ処理システムを提供することが知られている。そのようなシステムの例としてはクレイ１およびディジタルイクイップメント社のマルチタイタンプロセッサがある。

【０００３】

クレイ１プロセッサはベクトルレジスタバンクとスカラレジスタバンクを別々に有する。実行されている命令の演算コードがベクトル演算を示している場合、一連のデータ値は、長さレジスタに格納されている長さ値およびマスクレジスタに格納されているマスクに応じて、ベクトルレジスタバンクから戻される。長さはその一連のデータ値にいくつのデータ値があるかを指定し、マスクは命令に示されたベクトルレジスタに対応する複数のデータ値の中からどのデータ値が戻されるかを指定する。

【０００４】

マルチタイタンプロセッサは１つのレジスタバンクを有し、その中のレジスタはスカラまたはベクトルとして使用される。命令自体には、指定されたレジスタがスカラかまたはベクトルかを示すフラグと、ベクトルレジスタが使用されたときに前記一連のデータ値の中のデータ値の数を示す長さフィールドとが含まれる。

【０００５】

ベクトル命令自体は望ましい。なぜなら単一の命令で複数のデータ処理演算を指定できるのでコード密度を高めることができるからである。オーディオやグラフィックス処理等のデジタル信号処理はベクトル演算を利用するのに特に適している。なぜなら、一連の信号値に対してデジタルフィルタのタップ係数を掛けるフィルタ演算の実行等、一連の関連するデータ値に対して同じ演算を実行する要求が頻繁に発生するからである。

【０００６】

またデータ処理演算をできるだけ速く効率的に実行することが望ましい。速度と効率を上げる１つの方策は、レジスタバンク内にすでに格納されているデータ値を再度ロードまたは再度位置付ける必要を避けることである。これを実現する際、データ値を再使用できる命令コードは長くかつ複雑になりがちであるという問題がある。必要な演算を指示するのにより多くの命令が必要となれば、処理が遅くなりレジスタバンク内のデータ値を再使用するという目的が無効となる。

【０００７】

クレイ１やマルチタイタンプロセッサ等の汎用プロセッサを使用する代わりに、特殊目的のデジタル信号処理回路に少数のデジタル信号処理演算をサポートする特別な機能をもたせることがよく行われる。これらの特殊目的のデジタル信号処理回路内において、大きなメモリ内に必要なデータ値を格納し、必要に応じてそれぞれの演算に必要なデータ値を取り出すのが一般的な方法である。データ値は大きなメモリ内に再ロードしたり再位置付けする必要はない。なぜならそれらの使用順序は大きなメモリをアクセスするために使用したアドレスを操作することによって制御されるからである。この方法は、実行される演算に整合するように回路を特別に設計しなければならず、したがってより典型的な汎用プロセッサを使用した場合に得られる、他の機能との統合の柔軟性および容易性に欠けるという問題がある。

【０００８】

本発明の目的は、レジスタバンクとベクトル演算をサポートしている命令デコーダとを使用して、汎用プロセッサの柔軟性を維持しながら、効率的で速いデータ処理を提供することである。

【０００９】

１つの観点によれば、本発明によるデータ処理装置は、

複数のレジスタを有するレジスタバンクと、

前記レジスタバンク内の一連のレジスタのデータ値を使ってデータ処理演算を複数回実行するベクトル演算を指示する少なくとも１つのデータ処理命令に応動する命令デコーダとを備え、

前記レジスタバンクは少なくとも１つのレジスタサブセットを含み、前記一連のレジスタが前記サブセット内に存在し、

前記命令デコーダは前記一連のレジスタが前記レジスタサブセット内で回り込むように前記一連のレジスタを制御する、ことを特徴とする。

【００１０】

レジスタバンクの（すべてのレジスタ数よりも少ない）レジスタ数のレジスタサブセット内で回り込むようにしたので、データ値を再ロードまたは移動することなくレジスタバンク内のデータ値を再使用するコンパクトなコードを書くことができる。一連のベクトルレジスタを分割するための余分な命令を用いることなく必要な回り込みをハードウェアで行うことにより、命令コードは使用する度にサブセット内の異なった点から開始することができ、したがって異なった順序でデータ値を処理することができる。さらに、それら自身に対して回り込むレジスタのサブセットに対してベクトル演算を実行することにより、サブセット内に存在しない複数のレジスタのデータ値に対するデータ転送を同時に実行することができる。レジスタの回り込みはまた、例えば、バッファを互いにぐるぐる追いかけあう点において、データをバッファに取り込みバッファから乗算するようにしたリング（循環）バッファ型構成をサポートするハードウェアを提供することによっても得られる。

【００１１】

回り込みレジスタのサブセットは１つのみとすることもできるが、以下のようなシステムとする方が有利である。すなわち、前記ベクトル演算は、複数連のレジスタの対応する複数のデータ値を使って前記データ処理演算を実行し、

前記レジスタバンクは複数のレジスタサブセットを含み、前記複数連のレジスタがそれぞれのサブセット内に存在し、

前記命令デコーダは前記複数連のレジスタがそれぞれのレジスタサブセット内で回り込むように前記複数連のレジスタを制御する。

デジタル信号処理演算において、２連のレジスタからのデータ値を再使用する必要がしばしば発生する（例えば、タップおよび信号値をそれらのオフセットを異ならせて乗算・累算するＦＩＲ演算またはマトリックス演算）ので、複数の回

り込みレジスタサブセットを使用するのが望ましい。

【００１２】

サブセットが重なる可能性はあるが、実際にはそのような状況で再使用が必要なデータ値は普通離れているので、サブセットは互いに素としてもよい。これによって、ハードウェアのインプリメンテーションがより単純となり都合がよい。

【００１３】

複数のサブセットは、それらサブセット内に存在しないレジスタと混ざっている位置にあるレジスタで構成することもできる。しかし、複数サブセットが連続して番号付けられたレジスタのグループであれば、プログラミングおよびインプリメンテーションは容易になる。

【００１４】

レジスタグループはレジスタバンク内で互いに隔離することができるが、連続している方が好ましい。なぜならこの方が使用できるレジスタスペースをより効率的に利用できるからである。

【００１５】

ベクトル演算をより効果的に使用する本発明の能力は次の好ましい実施例において補完される。すなわち、メモリと、前記メモリと前記レジスタバンク内のレジスタとの間でのデータ値の転送を制御する転送コントローラと、をさらに有し、前記転送コントローラは複数の転送命令に応動して前記メモリと前記レジスタバンク内の一連のレジスタとの間で一連のデータ値を転送する、ことを特徴とする装置である。

【００１６】

レジスタバンク内のレジスタブロックに対してデータ値を受け渡しする能力により、レジスタブロックを数回再使用した後に１つの命令でそのブロックを取り替えることができるので、ベクトル演算を効率的に利用する本発明の能力が得られる。

【００１７】

ベクトル演算において一連のレジスタを使用することおよびレジスタバンクを前もって定められたレジスタサブセットに分割することは、各レジスタグループ

がそのレジスタグループの両端間で回り込みを行うインクリメンタを介してアドレスでアクセスされる、ことを特徴とする好ましい実施例において効率的に実施することができる。

【００１８】

ベクトル演算に使用される一連のレジスタは、サブセット内の１つおきのレジスタ等、多くの形態を取り得る。しかし、最も一般的に有用な形態は、一連のレジスタが連続した一連のレジスタであるような形態である。

【００１９】

上記の方法は、レジスタバンクを有しかつベクトル演算をサポートするいかなるプロセッサにも使用することができる。しかし、コードをコンパクトにしレジスタ内のデータ値を再使用する能力は、特に有用であることがわかっており、またレジスタバンクおよび命令デコーダが浮動小数点ユニット内に存在している実施例で採用された他の配慮に干渉しないことも分かっている。

【００２０】

別の観点によれば、本発明によるデータ処理方法は、

レジスタバンクの複数のレジスタ内にデータ値を格納するステップと、

ベクトル演算を指示する少なくとも１つのデータ処理命令に応動して、前記レジスタバンク内の一連のレジスタのデータ値を使ってデータ処理演算を複数回実行するステップを有するデータ処理方法であって、

前記レジスタバンクは少なくとも１つのレジスタサブセットを含み、前記一連のレジスタが前記サブセット内に存在し、

前記実行中に、前記一連のレジスタは前記レジスタサブセット内で回り込む、ことを特徴とする。

【００２１】

この方法は、ベクトル演算毎にタップ係数値と信号値との間の相対オフセットを異ならせてこれらの値を数回再使用するＦＩＲフィルタ演算を効率的に行う上で特に有用である。

本発明の実施例を１つの例として以下の図面を参照して説明する。

【００２２】

図１はデータ処理システム２２を示し、これはメインプロセッサ２４、浮動小数点ユニットコプロセッサ２６、キャッシュメモリ２８、メインメモリ３０、および入出力システム３２を含む。メインプロセッサ２４、キャッシュメモリ２８、メインメモリ３０および入出力システム３２はメインバス３４を介してつながっている。コプロセッサバス３６はメインプロセッサ２４を浮動小数点ユニットコプロセッサ２６に接続する。

【００２３】

動作に際して、メインプロセッサ２４（ＡＲＭコアとも称す）は、キャッシュメモリ２８、メインメモリ３０および入出力システム３２との対話を含む一般的なデータ処理演算を制御するデータ処理命令列を実行する。データ処理命令列内にはコプロセッサ命令が埋め込まれている。メインプロセッサ２４はこれらのコプロセッサ命令を、付属のコプロセッサによって実行すべきものとして認識する。したがって、メインプロセッサ２４はこれらのコプロセッサ命令をコプロセッサバス３６上に送ることにより、コプロセッサバス３６からコプロセッサ命令が付属のコプロセッサに受け取られる。浮動小数点ユニットコプロセッサ２６はそれ自身に向けられたものであることを検知したコプロセッサ命令を受け取り、実行する。この検知はコプロセッサ命令内部のコプロセッサ番号フィールドを介して行われる。

【００２４】

図２は浮動小数点ユニットコプロセッサ２６をより詳細に模式的に示すものである。浮動小数点ユニットコプロセッサ２６は、３２個の３２ビットレジスタ(図２には少く示してある)からなるレジスタバンク３８を含む。これらのレジスタは、各々が３２ビットデータ値を格納した単精度レジスタとして個別に動作するか、あるいは２つで６４ビットデータ値を格納した一対のレジスタとして動作することができる。浮動小数点ユニットコプロセッサ２６内には、パイプライン制御の乗累算ユニット４０とロードストア制御ユニット４２が設けられている。適切な状況において、パイプライン制御の乗累算ユニット４０とロードストア制御ユニット４２は同時に動作し、パイプライン制御の乗累算ユニット４０は算術演算（乗累算および他の演算を含む）をレジスタバンク３８内のデータ値に対し

て行い、ロードストア制御ユニット４２はパイプライン制御の乗累算ユニット４０が使っていないデータ値をメインプロセッサ２４を介して浮動小数点ユニットコプロセッサ２６に受け渡す。

【００２５】

浮動小数点ユニットコプロセッサ２６において、受け付けられたコプロセッサ命令は命令レジスタ４４内にラッチされる。この単純化された図において、コプロセッサ命令は演算コードとそれに続く３つのレジスタ指示フィールドＲ１、Ｒ２、Ｒ３（実際はこれらのフィールドは分割して命令全体に様々に分散させてもよい）から構成されると考えることができる。これらのレジスタ指示フィールドＲ１、Ｒ２、Ｒ３は、実行されているデータ処理演算の転送先、第１転送元および第２転送元として機能するレジスタバンク３８内のレジスタにそれぞれ対応している。ベクトル制御レジスタ４６（これは追加機能を行うより大きなレジスタの一部でもよい）は、浮動小数点ユニットコプロセッサ２６等で実行されるベクトル演算のための長さ値と歩幅値を格納する。ベクトル制御レジスタ４６はベクトル制御レジスタロード命令に応じて初期化して、長さ値と歩幅値で更新することができる。ベクトル長さ値と歩幅値は浮動小数点ユニットコプロセッサ２６内でグローバルに用いられるので、自己変更コードに頼ることなくこれらの値をグローバルベースで動的に変更することができる。

【００２６】

レジスタ制御・命令発行ユニット４８、ロードストア制御ユニット４２およびベクトル制御ユニット５０は共同して命令デコーダの機能の主要部分を実行すると考えることができる。レジスタ制御・命令発行ユニット４８は、演算コードと３つのレジスタ指示フィールドR１、Ｒ２、Ｒ３に応じて、演算コードに対してデコードしないであるいはベクトル制御ユニット５０を使用しないでレジスタバンク３８に初期レジスタアクセス(アドレス)信号を出力する。このように初期レジスタ値に直接アクセスできるため早い実行が達成される。ベクトルレジスタが指示されると、ベクトル制御ユニット５０は３ビットインクリメンタ(加算器)５２を使って一連のレジスタアクセス信号を作る。ベクトル制御ユニット５０は、ベクトル制御レジスタ４６内に格納された長さ値と歩幅値に対応してレジスタバン

ク３８にアドレスでアクセスする。レジスタスコアボード５４はレジスタロックを行うために設けられている。これはパイプライン制御の乗累算ユニット４０と同時に動作するロードストア制御ユニット４２がデータ整合性問題を起こさないようにするためである（レジスタスコアボード５４はレジスタ制御・命令発行ユニット４８の一部と考えることもできる）。

【００２７】

命令レジスタ４４内の演算コードは実行されるデータ処理演算の種別（命令が加算、減算、乗算、除算、ロード、ストア等のどれであるか）を指定する。これは指定されるレジスタがベクトルかスカラであるかに関係ない。これによって命令のデコードと乗累算ユニット４０のセットアップを容易にする。第１レジスタ指示値R１と第２レジスタ指示値R２は共同で、演算コードによって指定された演算のベクトル／スカラ種別をコード化する。コード化でサポートされる３つの一般的なケースは、S=S*S(例えば、Cコードのブロックからコンパイラによって作成される基本的ランダム計算)、V=V op S（例えば、ベクトルの要素を拡大縮小すること）、およびV=V op V(例えば、ＦＩＲフィルタとグラフィック変換等のマトリックス演算)（上記の文において、「op」とは一般的演算であり、構文は転送先＝第２オペランドop第１オペランドである）。また、命令によっては(例えば、ゼロまたは絶対値と比較する命令)、転送先レジスタを持たなかったり(例えば出力が条件フラグ)あるいは入力オペランドが足りない（ゼロと比較する命令がひとつの入力オペランドのみしか持たない）ことがある。このような場合、ベクトル／スカラ種別等のオプションを指定するためにより多くの演算コードビットスペースが使用可能であり、レジスタ全部を各オペランドに使用可能とすることができる(例えば、比較命令は常に、レジスタが何であれ完全にスカラであってもよい)。

【００２８】

レジスタ制御・命令発行ユニット４８とベクトル制御ユニット５０は命令デコーダの機能の主要部分を共同で実行するが、第１レジスタ指示値R１と第２レジスタ指示値R２に応じて、指示されたデータ処理演算のベクトル／スカラ種別を決定し制御する。ベクトル制御レジスタ４６内に格納された長さ値が１(格納さ

れたゼロの値に対応する）を示す場合、これは純粋スカラ演算の早期指示として用いることができる。

【００２９】

図３は流れ図であり、単精度モードにおいてレジスタ指示値からベクトル／スカラ種別を解読するために使う処理の流れを示す。ステップ５６において、ベクトル長さがグローバルに１（長さ値ゼロに相当する）に設定されているかを調べる。ベクトル長さが１である場合、ステップ５８においてすべてのレジスタはスカラとして扱われる。ステップ６０において、転送先レジスタR１がＳ０からＳ７の範囲内かどうかを調べる。そうである場合、演算はすべてスカラであり、ステップ６２に示すように、S=S op Sの構文形態となる。もしステップ６０がNOである場合、ステップ６４に示すように、転送先はベクトルであると決定される。転送先がベクトルの場合、コードは第２オペランドもベクトルとであるとして扱う。したがって、この段階において残る２つの可能性は、V=V op SおよびV=V op Vである。これら２つのオプションの区別は、第１オペランドがＳ０からＳ７のいずれかであるかどうかを調べるステップ６６のチェックによって行う。もしそうであれば演算はV=V op S、そうでなければV=V op Vである。これらの状態はそれぞれステップ６８と７０で認識される。

【００３０】

ベクトル長さが１に設定されている場合、レジスタバンク３８の３２個のレジスタはすべてスカラとして使用できる。なぜなら転送先に使用できるレジスタの数を制限することになるステップ６０のチェックに頼ることなく、演算のスカラ種別がステップ５８で識別されるからである。ステップ６０のチェックは、ベクトルとスカラ命令が組合わせて使われている場合、全スカラ命令を識別するのに有用である。また、ベクトルとスカラの混在モードで演算する場合、もし第１オペランドがスカラであるとすると、それはＳ０からＳ７のいずれかである。もし第１オペランドがベクトルであるなら、それはＳ８からＳ３１のいずれかであることになる。ベクトルである第１オペランドに対してレジスタバンク内で使用可能なレジスタ数の３倍を提供することは、一連のデータ値を保持するために必要なレジスタ数がベクトル演算を使用するとき一般的に大きいことに対する適応である

。

【００３１】

ユーザが実行したい一般的演算がグラフィック変換であることは理解されるであろう。一般的なケースにおいて、実行される変換は４＊４マトリックスで表すことができる。そのような計算にオペランドが再使用されるということは、ベクトルとして扱われるレジスタにマトリックス値を格納することが望ましいことを示している。同様に、入力ピクセル値は普通、再使用できるように、ベクトルとして扱うことができる４つのレジスタに格納される。マトリックス演算の出力は普通４つのレジスタに格納されたスカラ（別々のベクトルライン乗算を累積したもの）である。２倍の入力値と出力値を扱いたい場合、２４（１６＋４＋４）個のベクトルレジスタと８（４＋４）個のスカラレジスタが必要となる。

【００３２】

図４は図３に対応する流れ図であるが、この場合は倍精度モードを示す。前にも述べたように、倍精度モードにおいてはレジスタバンク３８内のレジスタスロットは一対として機能し、論理レジスタD０からD１５内の１６個の６４ビットデータ値を格納する。この場合、レジスタのベクトル／スカラ種別のコードは図３のそれから変更されている。すなわちステップ６０と６６のチェックは、ステップ７２と７４における「転送先はD０からD３のいずれかか？」および「第１オペランドはD０からD３のいずれかか？」というチェックにそれぞれ変わっている。

【００３３】

レジスタ指定フィールド内のレジスタのベクトル／スカラ種別を上述のようにコード化することは命令ビット空間を大幅に節約するけれども、減算や除算のような非変換演算の場合いくつかの困難を生じる。レジスタ構成がV=V op Sである場合、非変換演算の第１オペランドと第２オペランド間に対称性が欠如している問題については、追加命令によってレジスタ値を交換することなしに、命令セットを拡張することによって非可換演算の２つの異なるオペランドオプションを表すSUB、RSUBおよびDIV、RDIV等の演算コードのペアを取り入れることで克服できる。

【００３４】

図５はレジスタバンク３８のサブセット内にベクトルを回り込ませる方法を示す。特に単精度モードにおいて、レジスタバンクは４つのレジスタ範囲に分割され、それらのアドレスはＳ０からＳ７、Ｓ８からＳ１５、Ｓ１６からＳ２３およびＳ２４からＳ３１である。これらのレジスタ範囲は互いに共通要素を持たず連続している。図２において、８個のレジスタを有するこれらのサブセットの回り込み機能はベクトル制御ユニット５０内に３ビットインクリメンタ(加算器)５２を用いることによって提供できる。サブセット範囲を越えるとインクリメンタは後ろに回り込む。この単純な操作は、レジスタアドレススペース内の８ワードからなる複数範囲のサブセットを整合させることによって容易になる。

【００３５】

再び図５を参照する。レジスタの戻り動作の理解を助けるためにいくつかのベクトル演算が示されている。最初のベクトル演算は、開始レジスタがＳ２、ベクトル長さが４（ベクトル制御レジスタ４６内の長さ値である３によって示される）、および歩幅値が１（ベクトル制御レジスタ４６内の歩幅値であるゼロによって示される）を指定している。したがって、これらグローバルベクトル制御パラメータが設定された状態で命令がデコードされてレジスタＳ２をベクトルとして参照する場合、この命令はレジスタＳ２、Ｓ３、Ｓ４、Ｓ５内のデータ値をそれぞれ使って４回実行される。このベクトルはサブセット範囲を越えないので、ベクトルの回り込みは行われない。

【００３６】

第２の例では、開始レジスタがＳ１４、ベクトル長さが６、歩幅値が１である。この場合、命令はレジスタＳ１４から始まって６回実行されることになる。次に使用されるレジスタはＳ１５である。レジスタが歩幅値だけインクリメントすると、今度は使用されるレジスタがＳ１６ではなく、レジスタＳ８に回り込む。命令はさらに３回実行されて、全工程Ｓ１４、Ｓ１５、Ｓ８、Ｓ９、Ｓ１０およびＳ１１を完了する。

【００３７】

図５の最後の例では、開始レジスタがＳ２５、ベクトル長さが８、歩幅値が２である。最初に使用されるレジスタはＳ２５で、その次が歩幅値に従ってＳ２７、

Ｓ２９そしてＳ３１である。レジスタＳ３１を使用した後は、次のレジスタ値は後ろに回り込んでサブセットの始めに戻り、歩幅値２に従ってレジスタＳ２４を通過してレジスタＳ２５を使って演算を実行する。インクリメンタ５２は、ベクトルレジスタ間を移動する際、現在の値に歩幅値を加える３ビット加算器でよい。従って、歩幅は加算器に異なる歩幅値を供給することによって調整できる。

【００３８】

図６は倍精度モードにおいてレジスタバンク３８の回り込みを示す。このモードにおいて、レジスタのサブセットはＤ０からＤ３、Ｄ４からＤ７、Ｄ８からＤ１１およびＤ１２からＤ１５で構成される。倍精度モードにおけるインクリメンタ５２として機能する加算器に入力される最小値は２である。これは倍精度の歩幅である１に相当する。倍精度の歩幅が２の場合加算器には４を入力する必要がある。図６に示す最初の例では、開始レジスタがＤ０、ベクトル長さが４、歩幅値が１である。この場合、ベクトルレジスタの順番は、Ｄ０、Ｄ１、Ｄ２およびＤ３である。サブセットの範囲は越えていないので、この例では回り込みはない。第２の例では開始レジスタがＤ１５、ベクトル長さが２、歩幅値が２である。この場合ベクトルレジスタの順番はＤ１５およびＤ１３である。

【００３９】

図２において、ロードストア制御ユニット４２はその出力に５ビットインクリメンタを有し、ロード／ストア多重演算はベクトル演算に適用されるレジスタ回り込みは行われない。これによって単一のロード／ストア多重命令は必要な数の連続するレジスタをアクセスすることができる。

【００４０】

この回り込み構成を利用した演算の例として４つの信号値と４つのタップからなるユニットに分割したＦＩＲフィルタがある。シンタックスR8-R11 op R16-R19がR8opR16、 R9opR17、R10opR18およびR11opR19のベクトル演算を表している場合、ＦＩＲフィルタ演算は以下のように実行することができる。

８タップをR8-R15に、８信号値をR16-R23にロードせよ

R8-R11opR16-R19そして結果をR24-R27に入れよ

R9-R12opR16-R19そして結果をR24-R27に蓄積せよ

R10-R13opR16-R19そして結果をR24-R27に蓄積せよ

R11-R14opR16-R19そして結果をR24-R27に蓄積せよ

R8-R11に新しいタップを再度ロードせよ

R12-R15opR16-R19そして結果をR24-R27に蓄積せよ

R13-R8opR16-R19そして結果をR24-R27(R15->R8へ回り込む)に蓄積せよ

R14-R9opR16-R19そして結果をR24-R27(R15->R8へ回り込む)に蓄積せよ

R15-R10opR16-R19そして結果をR24-R27(R15->R8へ回り込む)に蓄積せよ

R12-R15に新しいタップを再度ロードせよ

タップがなくなったら、R16-R19に新しいデータを再びロードせよ

R12-R15opR20-R23そして結果をR28-R31に入れよ

R13-R8opR20-R23そして結果をR28-R31(R15->R8へ回り込む)に蓄積せよ

R14-R9opR20-R23そして結果をR28-R31(R15->R8へ回り込む)に蓄積せよ

R15-R10opR20-R23そして結果をR28-R31(R15->R8へ回り込む)に蓄積せよ

残りも上記と同様。

【００４１】

以上からロードは複数の累計から異なるレジスタに対するものであり、従って並列に行うことができる(すなわち、２重バッファリングが達成できる)。

【００４２】

図７Aはメインプロセッサ２４がコプロセッサ命令をどのように見るかを模式的に示す。メインプロセッサは命令内にあるビットの組合わせであるフィールド７６(これは分割してもよい)を使って命令をコプロセッサ命令として識別する。標準のARMプロセッサ命令セット内において、コプロセッサ命令はコプロセッサ

番号フィールド７８を含む。メインプロセッサにつながっているコプロセッサはこのコプロセッサ番号フィールド７８を使って特定のコプロセッサ命令がそれらコプロセッサに宛てられたものかどうかを調べる。ＤＳＰコプロセッサ(例えばＡＲＭによって作られるピッコロコプロセッサ)または浮動小数点ユニットコプロセッサ等の異なるタイプのコプロセッサに異なるコプロセッサ番号を割り当てることができ、従って同じコプロセッサバス３６を使って１つのシステム内で別々にアドレスでアクセスできる。コプロセッサ命令はまたコプロセッサが使用する演算コードと、３つの５ビットフィールドを含む。これら５ビットフィールドはそれぞれ、コプロセッサレジスタの中から宛先、第１オペランドおよび第２オペランドを指定する。コプロセッサロードやストア等いくつかの命令の場合、メインプロセッサは少なくとも部分的にコプロセッサ命令をデコードすることにより、コプロセッサとメインプロセッサが共に望ましいデータ処理演算を実行できるようにする。メインプロセッサはまた、そのような状況において実行する命令デコードの一環として、コプロセッサ番号内にコード化されたデータタイプをデコードすることもできる。

【００４３】

図７Ｂは、倍精度および単精度演算をサポートするコプロセッサが、受け取ったコプロセッサ命令を解読する場合を示す。そのようなコプロセッサには連続する２つのコプロセッサ番号が割り当てられ、コプロセッサ番号の最上位の３ビットを使ってそれ自身が宛先のコプロセッサであるかどうか確認する。このようにして、コプロセッサ番号の最下位のビットは宛先のコプロセッサを確認する上で余分なビットとなるので、これをそのコプロセッサ命令を実行する際に使用されるデータタイプを指定するために使うことができる。この例において、データタイプは、単精度または倍精度のデータサイズに対応する。

【００４４】

倍精度モードにおいてはレジスタの数は実質的に３２から１６に減少することがわかる。従って、レジスタフィールドサイズを小さくすることができるのであるが、その場合、どのレジスタを使うかのデコードはコプロセッサ命令内の既知の位置における自己包含フィールドから直接得ることはできず、コプロセッサ命

令の他の部分のデコードに依存する。これはコプロセッサの動作を複雑にするだけでなく、遅くしてしまうという不利益がある。コプロセッサ番号の最下位ビットを使用してデータタイプをコード化することで、演算コードが完全にデータタイプに依存しなくなり、そのデコードを単純化し速度を速めることになる。

【００４５】

図７Ｃは、図７Ｂのコプロセッサがサポートするデータタイプのサブセットである単一のデータタイプのみをサポートするコプロセッサがコプロセッサ命令を解読する場合を示す。この場合、完全なコプロセッサ番号を使ってその命令を受け付けるべきかどうかを決める。このように、もしコプロセッサ命令がサポートされていないデータタイプであれば、それは別のコプロセッサ番号に対応するので受け付けられない。そしてメインプロセッサ２４は未定義の命令例外処理を行い、サポートされていないデータタイプに対して演算をエミュレートする。

【００４６】

図８はＡＲＭコア８０を有するデータ処理システムを示す。ＡＲＭコア８０はメインプロセッサとして機能し、単精度および倍精度データタイプの両方をサポートするコプロセッサ８４にコプロセッサバス８２を介してつながっている。コプロセッサ番号を含むコプロセッサ命令は、命令列内で見出されたなら、ＡＲＭコア８０からコプロセッサバス８２上に渡される。そしてコプロセッサ８４はコプロセッサ番号をそれ自身の番号と比較し、一致したならＡＲＭコア８０に受付信号を送る。もし受付信号を受け取らなかったなら、ＡＲＭコアは未定義命令例外であることを認識し、メモリシステム８６に格納された例外処理コードを参照する。

【００４７】

図９は、図８のシステムにおけるコプロセッサ８４を単精度演算のみをサポートするコプロセッサ８８に取り替えたシステムを示す。この場合、コプロセッサ８８は１つのコプロセッサ番号のみを認識する。したがって、元の命令列内にある倍精度コプロセッサ命令は図８のコプロセッサ８４によっては実行されるが、単精度コプロセッサ８８によっては受け付けられない。したがって、同じコードを実行したい場合、メモリシステム８６内の未定義例外処理コードは倍精度エミ

ュレーションルーチンを含めることができる。

【００４８】

倍精度命令をエミュレートする必要はこれらの命令の実行を遅くするのではあるが、単精度コプロセッサ８８は倍精度コプロセッサ８４よりも小さく安価にでき、また倍精度命令が十分まれにしか現れない場合、有利である。

【００４９】

図１０は、単精度および倍精度命令の両方をサポートし２つの隣接するコプロセッサ番号を有するコプロセッサ８４内の命令ラッチ回路を示す。この場合、コプロセッサ命令内のコプロセッサ番号の最上位３ビットCP#[3:1]はコプロセッサ８４に割り当てられた番号と比較される。この例において、コプロセッサ８４がコプロセッサ番号１０と１１を持っている場合、この比較はコプロセッサ番号の最上位３ビットCP#[3:1]を２進数１０１に照らし合わせることで行うことができる。もし一致したなら、受付信号がARMコア８０に送り返され、コプロセッサ命令がラッチされて実行される。

【００５０】

図１１は、図９の単精度コプロセッサ８８内の等価回路を示す。この場合、１つのコプロセッサ番号のみが認識され、デフォールトとして単精度演算が使用される。コプロセッサ命令を受け付けてラッチすべきかどうかを決める上で行う比較は、コプロセッサ番号の全４ビットCP#[3:0]と埋め込まれた１つのコプロセッサ番号である二進数１０１０との間で行われる。

【００５１】

図１２は、図９例の未定義例外処理ルーチンを開始して倍精度エミュレーションコードを走らせる場合の流れを示す。そのために以下の手順を踏む。未定義命令例外を発生させる命令が、コプロセッサ番号である２進数１０１１を有するコプロセッサ命令であるかどうかを調べる(ステップ９０)。もしそうであれば、この命令は倍精度命令として意図されたものであり、ステップ９２でエミュレートできる。その後、メインプログラムの流れに戻る。他の例外タイプが、ステップ９０で検知されない場合、先のステップで検知し処理することもできる。

【００５２】

図１３は、レジスタバンク２２０の各３２ビットレジスタまたはデータスロットに格納されたデータのタイプを識別する情報を格納するためのフォーマットレジスタＦＰＲＥＧ２００を使用した例を示す。前に述べたように、各データスロットは３２ビットデータ値(データワード)を格納するための単精度レジスタとして個々に動作させることができる。あるいは、他のデータスロットと対にして６４ビットデータ値（２データワード）を格納するための倍精度レジスタとして動作させることもできる。本発明の好ましい実施例によれば、フォーマットレジスタＦＰＲＥＧ２００はデータスロットがその中に単精度データまたは倍精度データを格納しているかを識別するように構成されている。

【００５３】

図１３に示したように、レジスタバンク２２０の３２個のデータスロットは１６対のデータスロットを提供するように構成される。最初のデータスロットがその中に単精度データ値を格納している場合、好ましい実施例においてその対の他方のデータスロットは単精度データ値のみを格納し、倍精度データ値を格納するために他のデータスロットとリンクされることはない。これによって、どのデータスロット対も、２つの単精度データ値かあるいは１つの倍精度データ値のいずれかを確実に格納することになる。この情報はレジスタバンク２２０内の各データスロット対に関わる１ビット情報によって識別できる。したがって好ましい実施例において、フォーマットレジスタＦＰＲＥＧ２００はレジスタバンク２２０の各データスロット対に格納されるデータのタイプを識別する１６ビットの情報を格納するように構成される。したがってフォーマットレジスタＦＰＲＥＧ２００は１６ビットレジスタとして、あるいは浮動小数点ユニットコプロセッサ２６内の他のレジスタとの整合性を保つために、１６ビット情報を有する３２ビットレジスタとして構成することができる。

【００５４】

図１５はレジスタバンク２２０内の６対のデータスロットを示す。これらのデータスロット対は好ましい実施例によれば６個の倍精度データ値または１２個の単精度データ値を格納するために使うことができる。これらデータスロット内に格納できるデータの例を図１５に示す。ＤＨは倍精度データ値の３２個の最上位

ビットを表し、ＤＬは倍精度データ値の３２個の最下位ビットを表し、Sは単精度データ値を表す。

【００５５】

本発明の好ましい実施例によるフォーマットレジスタＦＰＲＥＧ２００内の対応するエントリも図１５に示す。好ましい実施例によれば、フォーマットレジスタＦＰＲＥＧ２００に格納された値「１」は対応するデータスロット対には倍精度データ値が格納されていることを示し、値「０」は対応するデータスロット対の少なくとも１つのデータスロットには単精度データ値が格納されており、あるいは両方のデータスロットが初期化されていないことを表している。つまり、両方のデータスロットが初期化されていないか、あるいはデータスロット対の内１つのデータスロットが初期化されておらず他方のデータスロットが単精度データ値を格納しているか、あるいはその対の両方のデータスロットが単精度データ値を格納している場合、ロジック「０」値がフォーマットレジスタＦＰＲＥＧ２００の対応するビットに格納される。

【００５６】

前に述べたように、好ましい実施例のFPUコプロセッサ２６を使用して、単精度データ値または倍精度データ値のいずれかを処理することができ、またメインプロセッサ２４が発行したコプロセッサ命令はそれが単精度命令かまたは倍精度命令(図７Ｂおよび関連する記述を参照)かを識別する。命令がコプロセッサに受け付けられた場合、レジスタ制御・命令発行ユニット４８に渡されてデコードおよび実行される。もし命令がロード命令であれば、レジスタ制御・命令発行ユニット４８はロードストア制御ユニット４２にメモリから特定されたデータを取り出して、それをレジスタバンク２２０の指定されたデータスロットに格納するよう指示する。この段階でコプロセッサは取り出されているデータが単精度データ値か倍精度データ値かを知ることになり、ロードストア制御ユニット４２はそれに応じて動作する。したがって、ロードストア制御ユニット４２は３２ビット単精度データ値または６４ビット倍精度データ値のいずれかを経路２２５を介してレジスタバンク入力ロジック２３０に渡してレジスタバンク２２０に格納させる。

【００５７】

データはロードストア制御ユニット４２によってレジスタバンク２２０にロードされるだけでなく、フォーマットレジスタＦＰＲＥＧ２００にも供給されて必要なビットをそれに追加して、データを受け取った各データスロット対が単精度データまたは倍精度データを格納しているのかを識別できるようにする。好ましい実施例において、このデータはレジスタバンクにロードされる前にフォーマットレジスタＦＰＲＥＧ２００に格納される。これはこの情報をレジスタバンク入力ロジック２３０が使えるようにするためである。

【００５８】

好ましい実施例において、レジスタバンク２２０内のデータの内部フォーマットは外部フォーマットと同じである。したがって単精度データ値は３２ビットデータ値として、また倍精度データ値は６４ビットデータ値としてレジスタバンク２２０内に格納される。レジスタバンク入力ロジック２３０はフォーマットレジスタＦＰＲＥＧ２００にアクセスできるので、それが現在受け取っているデータが単精度か倍精度かを知っている。したがってそのような実施例においてレジスタバンク入力ロジック２３０は経路２２５を介して受け取ったデータをレジスタバンク２２０の適切なデータスロットに格納するようにデータを配列するだけである。しかし、別の実施例において、レジスタバンク内の内部フォーマットが外部フォーマットと異なる場合、レジスタバンク入力ロジック２３０は必要な変換を行うように構成することもできる。例えば、数値は普通１．ａｂｃ…にベース値を指数だけべき乗した値を掛けたものとして表す。効率のために、典型的な単精度と倍精度表現は小数点の左側の１を表すためにデータビットを使わない。むしろその１は暗に含まれている。レジスタバンク２２０内で使われている内部表現が何らかの理由で１を明示的に表す必要がある場合、レジスタバンク入力ロジック２３０は必要なデータ変換を行う。そのような実施例において、レジスタバンク入力ロジック２３０によって作られた追加データを収容するためにデータスロットは普通３２ビットよりいくらか大きい。

【００５９】

ロードストア制御ユニット４２は、データ値をレジスタバンク２２０にロード

するだけでなく、データをコプロセッサ２６の１つまたは複数のシステムレジスタ、例えばユーザステータス・制御レジスタＦＰＳＣＲ２１０、にロードすることもできる。好ましい実施例において、ユーザステータス・制御レジスタＦＰＳＣＲ２１０はユーザがアクセスできる構成ビットおよび例外ステータスビットを含む。この詳細は実施例の説明の終わりに示した浮動小数点ユニットのアーキテクチャの説明にゆだねる。

【００６０】

レジスタ制御・命令発行ユニット４８がストア命令を受け取り、その命令がメモリに格納すべき内容を有するレジスタバンク２２０内のデータスロットを特定している場合、ロードストア制御ユニット４２はそれに応じた動作を指示され、必要なデータワードがレジスタバンク２２０からレジスタバンク出力ロジック２４０を介してロードストア制御ユニット４２に読み出される。レジスタバンク出力ロジック２４０は、読み出されつつあるデータが単精度データか倍精度データかを判断するためにＦＰＲＥＧレジスタ２００の内容にアクセスする。そして適切なデータ変換を行うことによりレジスタバンク入力ロジック２３０が行ったデータ変換を元に戻し、そのデータを経路２３５を介してロードストア制御ロジック４２に供給する。

【００６１】

本発明の好ましい実施例によれば、ストア命令が倍精度命令である場合、コプロセッサ２６は命令を倍精度データ値に適用する第２の動作モードで動作していると考えることができる。倍精度データ値は偶数個のデータワードを保持しているので、第２の動作モードで発行されたストア命令は普通、メモリに格納すべき内容を有する偶数個のデータスロットを特定する。しかし本発明の好ましい実施例によれば、もし奇数個のデータスロットが指定されると、ロードストア制御ユニット４２はＦＰＲＥＧレジスタ２００の内容を読み、まずこれらの内容をメモリに格納してからレジスタバンク２２０内の特定された偶数個のデータスロットを格納する。普通転送すべきデータスロットを特定するには、まずベースアドレスでレジスタバンク内の特定データスロットを指定し、次に指定されたデータスロットから始まる格納すべきデータスロットの数(すなわちデータワードの数)を

数値で指定する。

【００６２】

したがって、例えばストア命令がベースアドレスとしてレジスタバンク２２０内の最初のデータスロットを与え、３３個のデータスロットを指定した場合、全３２個のデータスロットの内容がメモリに格納されるが、指定されたデータスロットの数が奇数なので、ＦＰＲＥＧレジスタ２００の内容もメモリに格納される。

【００６３】

この方法によって、レジスタバンクの内容と、レジスタバンク２２０のデータスロット内に格納されたデータタイプを識別するＦＰＲＥＧレジスタ２００の内容の両方を、１つの命令を使ってメモリに格納することができる。これによってＦＰＲＥＧレジスタ２００の内容を明示的に格納するために別の命令を発行しなければならないという問題を回避でき、したがってメモリにストアまたはメモリからロードの命令実行中に処理スピードに悪影響を及ぼすことがない。

【００６４】

本発明の更なる実施例において、この手法をさらに一段階進めることによって、追加のシステムレジスタ、例えばＦＰＳＣＲレジスタ２１０、を必要に応じて１つの命令を使ってメモリに格納することもできる。前に検討した３２個のデータスロットを有するレジスタバンク２２０の例を考えてみる。３３個のデータスロットがストア命令で特定された場合、ＦＰＲＥＧレジスタ２００はレジスタバンク２２０内の３２個のデータスロットの内容と共にメモリに格納される。しかし、レジスタバンク内のデータスロット数を越える異なる奇数、例えば３５、が特定された場合、ロードストア制御ユニット４２はこれを、ＦＰＲＥＧレジスタ２００の内容とレジスタバンク２２０のデータスロットだけでなくＦＰＳＣＲレジスタ２１０の内容もメモリに格納すべき要求と解釈することができる。コプロセッサはさらなるシステムレジスタ、例えばコプロセッサによって命令を処理中に発生した例外を特定する例外レジスタ、を設けることもできる。ストア命令において異なる奇数、例えば３７、が特定された場合、ロードストア制御ユニット４２はこれを、ＦＰＳＣＲレジスタ２１０、ＦＰＲＥＧレジスタ２００とレジス

タバンク２２０の内容と共に１つまたは複数の例外レジスタの内容も格納すべき要求と解釈することができる。

【００６５】

この手法は、ストアまたはロード命令を指示するコードがレジスタバンクの内容を知らない場合や、レジスタバンク内容がほんの一時的にメモリに格納されてレジスタバンクに後で取り出される場合に特に有用である。コードがレジスタバンクの内容を知っている場合、ＦＰＲＥＧレジスタ２００の内容をメモリに格納する必要はないかもしれない。レジスタバンクの内容を知らないコードの代表例はコンテクスト切替コード、および手順呼び出しエントリおよび出口ルーチンである。

【００６６】

そのような場合、ＦＰＲＥＧレジスタ２００の内容はレジスタバンクの内容と共に効率的にメモリに格納でき、また上で述べたように確かに他のシステムレジスタも必要に応じて格納することができる。

【００６７】

後続のロード命令を受け取ると、同様なプロセスが行われる。したがって、ロードストア制御ユニット４２は奇数個のデータスロットを指定する倍精度ロード命令を受け取ると、ＦＰＲＥＧレジスタ２００の内容をＦＰＲＥＧレジスタ２００にロードし、ロード命令で特定されたスロット数で示されるシステムレジスタの内容をロードし、さらに偶数個のデータワードをレジスタバンク２２０の指定されたデータスロットに格納する。したがって、前に述べた例で考えると、ロード命令で指定されたデータスロット数が３３である場合、ＦＰＲＥＧレジスタ内容はＦＰＲＥＧレジスタ２００にロードされ、それに続いて３２個のデータスロットの内容がロードされる。同様にロード命令で指定されたデータスロット数が３５である場合、上記の内容だけでなく、ＦＰＳＣＲレジスタ２１０の内容もＦＰＳＣＲレジスタにロードされる。最後に、指定されたデータスロット数が３７である場合、上記の内容だけでなく、例外レジスタの内容もそれら例外レジスタにロードされる。特定の奇数に関わる特定の動作はまったく任意であり、必要に応じて変更可能であることは当業者にとって明らかであろう。

【００６８】

図１４は、ストア命令およびロード命令を実行する時の、本発明の好ましい実施例によるレジスタ制御・命令発行ロジック４８の動作を示す流れ図である。最初に、ステップ３００において命令で特定された最初のレジスタ番号、すなわちベースレジスタと共に、データワード数(これは好ましい実施例においてデータスロット数と同じ)を命令から読み出す。そしてステップ３１０において、この命令が倍精度命令であるかどうかを調べる。前に述べたように、この情報はこの段階でコプロセッサに入手可能である。なぜなら命令はそれが倍精度命令か単精度命令かを特定しているからである。

【００６９】

命令が倍精度命令である場合、処理はステップ３２０へ進み、そこで命令で指定されたワード数が奇数かどうかを判断する。本実施例においては、ＦＰＲＥＧレジスタ２００と共にシステムレジスタを選択的に転送するための技法は使用しないと仮定して、ワード数が奇数である場合、これはＦＰＲＥＧレジスタ２００の内容を転送すべきことを示しており、したがってステップ３２５において、ＦＰＲＥＧレジスタの内容がロードストア制御ユニット４２によって転送される。そして、ステップ３２７でワード数が１だけインクリメントされ、処理はステップ３３０に進む。もしステップ３２０においてワード数が偶数であると判断されたなら、処理は直接ステップ３３０に進む。

【００７０】

ステップ３３０において、ワード数がゼロよりも大きいかどうか判断される。そうでない場合、命令は完了したとみなされ、ステップ３４０で処理を抜け出す。しかしワード数がゼロよりも大きい場合、処理はステップ３３２に進んで倍精度データ値(すなわち２つのデータスロットの内容)が最初に指示されたレジスタ番号へ受け渡される。その後、ステップ３３４においてワード数が２だけデクリメントされ、ステップ３３６においてレジスタ番号が１だけインクリメントされる。前に述べたように、倍精度命令の場合、レジスタは実際２つのデータスロットからなるので、レジスタカウントを１だけインクリメントすることはデータスロット数を２だけインクリメントすることと同じである。

【００７１】

そして処理はステップ３３０に戻り、そこでワード数がまだゼロより大きいかどうか判断する。まだ大きい場合、処理は繰り返される。ワード数がゼロになったら、ステップ３４０で処理を抜け出る。

【００７２】

ステップ３１０において、命令が倍精度命令ではないと判断された場合、処理はステップ３５０に進んで再びワード数がゼロよりも大きいかどうか判断する。大きい場合、ステップ３５２に進んで、命令で指定された最初のレジスタ番号へ単精度データ値を受け渡す。そしてステップ３５４においてワード数を１だけデクリメントし、ステップ３５６においてレジスタカウントを１だけインクリメントして次のデータスロットを指示する。そして処理はステップ３５０に戻り、ワード数がまだゼロより大きいかどうか判断する。大きい場合、ワード数がゼロになるまで処理を繰り返す。ゼロになったらステップ３６０で処理を抜け出る。

【００７３】

上記の方法は、例えばコンテクスト切替コード、手順呼び出しエントリや出口ルーチン等、レジスタバンクの内容を知らないコードを実行する際に大きな融通性を与える。これらの場合、オペレーティングシステムはレジスタの内容を知らないので、それらの内容によってレジスタを異なった扱いをしなくてすむようにすることが望ましい。上記の方法によれば、奇数個のデータワードを指定する１つのストアまたはロード命令でこれらのコードルーチンを書き込むことができる。コプロセッサがレジスタ内容の情報を使う必要がある場合、命令に奇数個のデータワードが指定されているとき、コプロセッサはこれをレジスタバンク内のデータの内容を特定するために必要なフォーマット情報もメモリへ格納またはメモリからロードする要求であると解釈する。この柔軟性により、レジスタ内容の情報を必要とするコプロセッサをサポートするためにユニークなオペレーティングシステムソフトウェアを使う必要性がなくなる。

【００７４】

この方法はまた、コード内で別の演算によってレジスタ内容の情報をロードしたり格納したりする必要性を解消する。レジスタ内容の情報をロードしたりスト

アする選択肢が命令の中に組み込まれているため、追加のメモリアクセスは必要ない。これによりコード長さが短くなり、潜在的に時間が節約できる。

【００７５】

上記の技法を組み込んだ浮動小数点ユニットのアーキテクチャに関する説明を以下に示す。

【００７６】

１．　序論

ＶＦＰｖ１はＡＲＭプロセッサモジュールに使用するコプロセッサとして実現するように設計された浮動小数点システム（ＦＰＳ）のアーキテクチャである。このアーキテクチャを実施するとハードウェアまたはソフトウェアの特徴的機能を組み込むことができる。あるいはソフトウェアを使って機能の補充やＩＥＥＥ７５４互換性を提供することができる。この仕様書はハードウェアおよびソフトウェアサポートを使って完全なＩＥＥＥ７５４互換性を達成するよう意図されている。

【００７７】

２つのコプロセッサ番号がＶＦＰｖ１に使用されている。１０は単精度オペランドの演算に使用され、１１は倍精度オペランドの演算に使われる。単精度データと倍精度データ間の変換は、ソースオペランドコプロセッサスペースで動作する２つの変換命令で達成される。

【００７８】

ＶＦＰｖ１アーキテクチャの特徴は以下のとおり。

・　ハードウェアにおいてサポートコードを使ってＩＥＥＥ７５４と完全な互換性がある

・　３２個の単精度レジスタ。各々がソースオペランドまたは転送先レジスタとしてアドレス指定可能

・　１６個の倍精度レジスタ。各々がソースオペランドまたは転送先レジスタとしてアドレス指定可能。（倍精度レジスタは物理的単精度レジスタと重なる）

・　ベクトルモードは浮動小数点コード密度とロードおよびストア動作の同時実行性を大幅に高める。

・　８個の循環単精度レジスタからなる４つのバンク、あるいは４個の倍精度レジスタからなる４つのバンクがＤＳＰおよびグラフィック演算を高める

・　非正規処理オプションが（浮動小数点エミュレーションパッケージからのサポートを前提に）ＩＥＥＥ７５４互換性あるいは高速ゼロクリア能力のいずれかを選択する

・　ＩＥＥＥ７５４互換性のある結果を出す、完全にパイプランされてつながれた乗累算の実現

・　ＦＦＴＯＳＩＺ命令を使ってＣ、Ｃ＋＋およびＪａｖａ用に浮動小数点から整数へ高速変換

【００７９】

作成者は、完全にハードウェアでVFPｖ１を実現することもできるし、ハードウェアとサポートコードを組合わせて利用することもできる。VFPｖ１は完全にソフトウェアで実現することもできる。

【００８０】

２．技術用語

本仕様書は以下の用語を使用している。

Automatic exception （自動例外）：　例外的状態である。これはそれぞれの例外イネーブルビットの値に関わらずこの例外的状態からサポートコードに飛ぶ。どの例外を自動にするかの選択は作成時点でのオプションである。セクション０を参照。

【００８１】

例外処理

【００８２】

Ｂｏｕｎｃｅ（バウンス）：オペレーティングシステムに報告された例外であって、ユーザトラップハンドラを呼び出すことなくあるいはユーザコードの正常な流れを遮ることなくサポートコードによって完全に処理される。

【００８３】

CDP:'Coprocessor Data Processing'FPS(浮動小数点システム)の場合、ＣＤＰ処理はロードまたはストア演算ではなく算術演算である。

【００８４】

ConvertToUnsignedInteger(Fm):Fm内の内容を符号なし３２ビット整数値に変換すること。結果は、３２ビット符号なし整数の範囲外の浮動小数点値の最終丸めおよび取扱いのための丸めモードに依存する。ＩＮＶＡＬＩＤ例外は浮動小数点入力値が負の値であるか、３２ビット符号なし整数としては大きすぎる場合に起こる可能性がある。

【００８５】

ConverToSignedInteger(Fm):Fmの内容を符号付３２ビット整数値に変換すること。結果は、３２ビット符号付整数の範囲外の浮動小数点値の最終丸めおよび取扱いのための丸めモードに依存する。ＩＮＶＡＬＩＤ例外は浮動小数点入力値が３２ビット符号付整数としては大きすぎる場合に起こる可能性がある。

【００８６】

ConvertUnsignedIntToSingle/Double(Rd):３２ビット符号なし整数値として解読されるARMレジスタ（Ｒｄ）の内容を単精度または倍精度浮動小数点値に変換すること。転送先精度が単精度である場合、変換演算でINEXACT例外が発生する可能性がある。

【００８７】

ConvertSignedIntToSingle/Double(Rd):３２ビット符号なし整数値として解読されるＡＲＭレジスタ（Ｒｄ）の内容を単精度または倍精度浮動小数点値に変換すること。転送先精度が単精度である場合、変換演算でＩＮＥＸＡＣＴ例外が発生する可能性がある。

【００８８】

Denormalized value(非正規化された値):($-2^{Emin} < x < 2^{Emin}$)の範囲の値を表す。単精度および倍精度オペランドのＩＥＥＥ７５４フォーマットにおいて、非正規化値は指数がゼロで、先頭ビットは１ではなく０である。ＩＥＥＥ７５４－１９８５仕様は、非正規化オペランドの作成および操作は正規化オペランドと同じ精度で行わなければならないと規定している。

【００８９】

Disabled exception(禁止された例外):FPCSR内の関連する例外イネーブルビッ

トが０に設定されている場合、例外は’禁止されている’という。これらの例外に対して、ＩＥＥＥ７５４仕様は戻されるべき正しい結果を定義している。例外条件を発生する演算はサポートコードへバウンスしてＩＥＥＥ７５４で定義された結果を生み出す。例外はユーザ例外ハンドラへは報告されない。

【００９０】

Enabled exception(許可された例外):関連する例外イネーブルビットが１に設定された例外。この例外が発生した場合、トラップがユーザハンドラに掛けられる。例外条件を発生する演算はサポートコードへバウンスしてＩＥＥＥ７５４で定義された結果を生み出す。例外はユーザ例外ハンドラに報告される。

【００９１】

Ｅｘｐｏｎｅｎｔ（指数）:表した数字の値を確定するために２をべき乗するときの整数ベキ数を表す浮動小数点数のコンポーネントである。たまに指数は符号付または符号なし指数と呼ばれる。

【００９２】

Fraction（小数）:暗示される２進小数点の右側にあるsignificand(有効数字)のフィールド

【００９３】

Flush-To-Zero Mode:このモードでは、丸めた後に($-2^{Emin} < x < 2^{Emin}$)の範囲にあるすべての値は、非正規化値に変換されるのではなくゼロとして扱われる。

【００９４】

High(Fn/Fm):メモリ内で表された倍精度値の上位３２ビット[63:32]

【００９５】

IEEE754-1985:"IEEE Standard for Binary Floating-Point Arithmetic", ANSI/IEEE Std 754-1985, The Institute of Electrical and Electronics Engineers, Inc. New York, New York, 10017.しばしばIEEE７５４標準と呼ばれるが、この標準は浮動小数点システムにおけるデータタイプ、正しい演算、例外タイプと取扱い、および誤り限度を定義する。ほとんどのプロセッサは、ハードウェアにおいてあるいはハードウェアとソフトウェアの組合わせにおいて本標準に準拠するように作られる。

【００９６】

Infinity:IEEE７５４の特殊フォーマットで無限大∞を表す。指数は、精度およびｓｉｇｎｉｆｉｃａｎｄ（有効数字）が全てゼロなので最大となる。

【００９７】

Input exception:与えられた演算のための１つまたは複数のオペランドがハードウェアでサポートされていない例外条件。演算は演算を完了するためにサポートコードにバウンスする。

【００９８】

Intermediate result(中間結果):丸める前に計算結果を格納するために用いる内部フォーマット。このフォーマットは転送先フォーマットよりも大きな指数フィールドとsignificand(有効数字)フィールドを有することもある。

【００９９】

Low(Fn/Fm):メモリ内で表された倍精度値の下位３２ビット[31:0]

【０１００】

MCR:”ＡＲＭレジスタからコプロセッサへ移動”。ＦＰＳの場合、これにはＡＲＭレジスタとＦＰＳレジスタとの間でデータまたは制御レジスタを転送する命令が含まれる。１つのＭＣＲクラス命令を使って３２ビットの情報のみを転送することができる。

【０１０１】

ＭＲＣ：”コプロセッサからＡＲＭレジスタへ移動”。ＦＰＳの場合、これにはＦＰＳとＡＲＭレジスタとの間でデータまたは制御レジスタを転送する命令が含まれる。１つのＭＲＣクラス命令を使って３２ビットの情報のみを転送することができる。

【０１０２】

ＮａＮ:数字ではなく、浮動小数点フォーマットでコード化された記号的実体。ＮａＮには２つのタイプがある：信号および非信号または静止。信号ＮａＮは、オペランドとして使用する場合Invalid Operand(無効オペランド)例外を生じる。静止ＮａＮは、例外を知らせることなくほとんどすべての算術演算に伝播する。NaNのフォーマットは指数フィールドがすべて１のゼロでないsignificand(

有効数字)である。信号NaNを表すには、小数の最上位ビットがゼロで、静止ＮａＮは最上位ビットが１に設定される。

【０１０３】

Reserved(予約済み):フィールドがインプリメンテーションによって定義されることになっている場合、制御レジスタまたは命令フォーマットのフィールドは"予約済み"となり、フィールドの内容がゼロでないならUNPREDICTABLE(予測不可能)な結果を生じる。これらのフィールドは将来のアーキテクチャ拡張時に使用するために予約されるか、あるいは特定のインプリメンテーションにのみ使用される。インプリメンテーションで使用されないすべての予約済みビットはゼロと書き込まれなければならずゼロとして読み出される。

【０１０４】

Rounding Mode(丸めモード)：ＩＥＥＥ７５４仕様は、すべての計算が無限の精度を有するかのように実行されるよう要求している。すなわち２つの単精度値の乗算はsignificand(有効数字)のビット数の２倍の精度でsignificand(有効数字)を計算しなければならない。この値を転送先の精度で表すには、significand(有効数字)を丸める必要がしばしばでてくる。ＩＥＥＥ７５４標準は４つの丸めモードを指定している：四捨五入する（ＲＮ）モード、ゼロに丸めまたは切り捨てる（ＲＺ）モード、正の無限大に丸める（ＲＰ）モード、そして負の無限大に丸める（ＲＭ）モードである。最初のモードは中間点で丸め、タイケースのときsignificand(有効数字)の最下位ビットがゼロになるのであれば切り上げて数字を「偶数」にする。２番目のモードはsignificand(有効数字)の右側のビットを実質的に常に切り捨てる。これは整数変換においてＣ、Ｃ＋＋、およびＪａｖａ言語で使用される。後の２つのモードは区間演算において使用される。

【０１０５】

Significand(有効数字)：２進浮動小数点数のコンポーネントであり、その暗示された２進小数点の左側にある明示あるいは暗示された先頭ビットと、右側にある小数フィールドとからなる。

【０１０６】

Support Code（サポートコード）：ＩＥＥＥ７５４標準との互換性を与えるた

めにハードウェアを補充する際に使用しなければならないソフトウェアである。サポートコードは２つのコンポーネントを有する。１つはルーチンの集まりで、超越計算等ハードウェアの範囲を越えた演算を実行し、また例外を発生させる可能性のあるサポートされていない入力を使っての除算等、サポートされている機能を実行するものである。もう１つは例外ハンドラの集合で、ＩＥＥＥ７５４に準拠するように例外条件を処理するものである。サポートコードは作成された機能を実行して、サポートされていないデータタイプまたはデータ表現(例えば、非正規値または１０進データタイプ)の適切な取扱いをエミュレートしなければならない。ルーチンの出口においてユーザの状態を回復させるように注意を払うならば、ルーチンは中間計算においてＦＰＳを利用するように書くこともできる。

【０１０７】

Ｔｒａｐ(トラップ)：それぞれの例外イネーブルビットをＦＰＳＣＲにセットする例外条件である。ユーザのトラップハンドラが実行される。

【０１０８】

UNDEFINED(未定義)：未定義命令トラップを発生させる命令のことである。ＡＲＭ例外に関する詳しい情報についてはARM Architectural Reference Manualを参照のこと。

【０１０９】

UNPREDICTABLE（予測不可能）：信頼できない、命令の結果または制御レジスタフィールド値である。予測不可能な命令や結果が機密上の欠点になったり、プロセッサあるいはシステムのいかなる部分をも停止させるようなことがあってはならない。

【０１１０】

Unsupported Data（サポートされていないデータ）：ハードウェアでは処理されないで、サポートコードにバウンスされて処理される特定データ値。これらのデータには、無限、ＮａＮ、非正規値、およびゼロがある。これらの値の内どれをハードウェアで完全にまたは部分的にサポートするか、あるいは演算処理を完了するためにサポートコードの助けを必要とするかを自由にインプリメンテーシ

ョンで選択できる。サポートされていないデータを処理したことから生じる例外は、対応する例外イネーブルビットがセットされているならば、ユーザコードへトラップされる。

【０１１１】

３．　レジスタファイル

３．１　序説

アーキテクチャは３２個の単精度レジスタと１６個の倍精度レジスタを備え、すべてのレジスタは転送元または転送先オペランドとして、完全に定義された５ビットレジスタインデックス内で個別にアドレス指定可能である。

【０１１２】

３２個の単精度レジスタは１６個の倍精度レジスタと重なっている、すなわち倍精度データをD５へ書き込むとS１０とS１１の内容を上書きする。コンパイラまたはアセンブリ言語プログラマは、レジスタを重なり合ったインプリメンテーションで使用する場合、レジスタを単精度データ記憶域として使用することと倍精度データ記憶域の半分として使用することが競合することを知らなければならない。レジスタの使用を一方の精度に制限するためのハードウェアは設けられていないので、このことを守らなければ結果は予測不可能となる。

【０１１３】

ＶＦＰｖ１はスカラモードまたはベクトルモードでこれらのレジスタにアクセスできるようにしている。スカラモードにおいては１つ、２つまたは３つのオペランドレジスタを使って生じた結果が転送先レジスタに書き込まれる。またベクトルモードにおいては、指定されたオペランドがレジスタのグループを参照する。ＶＦＰｖ１は、単精度オペランドの場合１つの命令において最大８個の要素に対して、また倍精度オペランドの場合最大４つの要素に対してベクトル演算をサポートしている。

表１　ＬＥＮビットコード化

LEN	ベクトル長コード化
000	スカラ
001	ベクトル長２
010	ベクトル長３
011	ベクトル長４
100	ベクトル長５
101	ベクトル長６
110	ベクトル長７
111	ベクトル長８

【０１１４】

ベクトルモードはゼロ以外の値をＬＥＮフィールドに書き込むことによって許可される。ＬＥＮフィールドに０が入っていると、ＦＰＳはスカラモードで動作し、レジスタフィールドは、フラットレジスタモデルにおいて３２個の単精度レジスタまたは１６個の倍精度レジスタをアドレス指定することと解読される。ＬＥＮフィールドがゼロでない場合、ＦＰＳはベクトルモードで動作し、レジスタフィールドはレジスタのアドレスベクトルとして動作する。ＬＥＮフィールドのコードについては表１を参照のこと。

【０１１５】

ＬＥＮフィールドを変えることなくスカラとベクトル演算を混在させる方法は転送先レジスタを指定することで行える。スカラ演算は、転送先レジスタが第１レジスタバンク（Ｓ０－Ｓ７またはＤ０－Ｄ３）にあるならベクトルモードにおいて指定することができる。詳細についてはセクション０を参照。

【０１１６】

３．２　単精度レジスタの使用方法

ＦＰＳＣＲ内のＬＥＮフィールドが０の場合、３２個の単精度レジスタＳ０－Ｓ３１が使用できる。これらレジスタのいずれでも転送元または転送先レジスタ

として使うことができる。

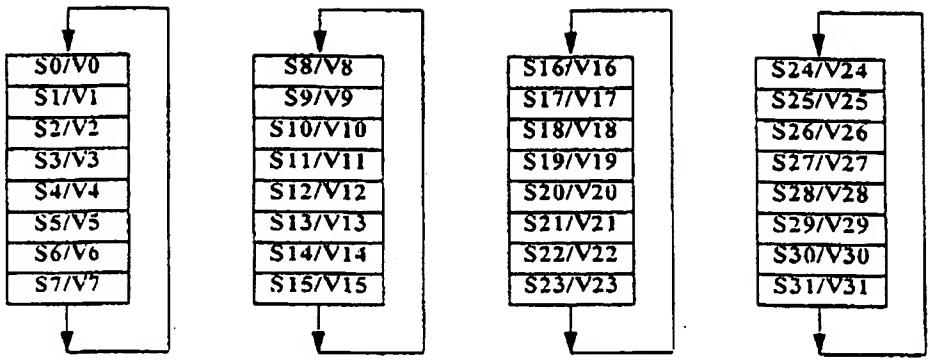
31	31	31	31 0
S0	S8	S16	S24
S1	S9	S17	S25
S2	S10	S18	S26
S3	S11	S19	S27
S4	S12	S20	S28
S5	S13	S21	S29
S6	S14	S22	S30
S7	S15	S23	S31

図例１　単精度レジスタマップ

単精度（コプロセッサ１０）レジスタマップは図例１に示すように作成することができる。

【０１１７】

ＦＰＳＣＲ内のＬＥＮフィールドが０よりも大きい場合、レジスタファイルは図２に示すようにそれぞれが８個の循環レジスタからなる４つのバンクとして挙動する。ベクトルレジスタの第１バンクＶ０－Ｖ７はスカラレジスタＳ０－Ｓ７にオーバラップし、各オペランドに対して選択されたレジスタによってスカラまたはベクトルとしてアドレスでアクセスされる。詳細はセクション０、３．４レジスタの使用法を参照。



図例２　循環単精度レジスタ

【０１１８】

例えば、もしＦＰＳＣＲ内のＬＥＮが３にセットされていれば、ベクトルＶ１０を参照するとレジスタＳ１０、Ｓ１１、Ｓ１２およびＳ１３がベクトル演算に関わってくる。同様に、Ｖ２２を参照するとＳ２２、Ｓ２３、Ｓ１６およびＳ１７が演算に関わってくる。レジスタファイルがベクトルモードでアクセスされると、Ｖ７に続くレジスタはＶ０である。同様にＶ８はＶ１５の後に続き、Ｖ１６はＶ２３に続き、そしてＶ２４はＶ３１に続く。

【０１１９】

３．３　倍精度レジスタの使用法

ＦＰＳＣＲ内のＬＥＮフィールドが０の場合、１６個の倍精度スカラレジスタが使用できる。

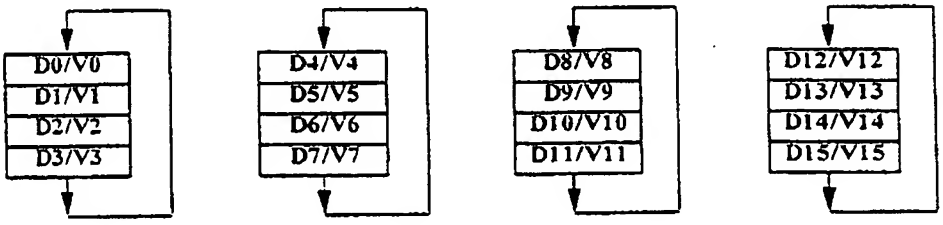
63	0 63	0
D0	D8	
D1	D9	
D2	D10	
D3	D11	
D4	D12	
D5	D13	
D6	D14	
D7	D15	

図例３　倍精度レジスタマップ

いずれのレジスタも転送元あるいは転送先レジスタとして使用できる。レジスタマップは図例３に示すように作成できる。

【０１２０】

ＦＰＳＣＲ内のＬＥＮフィールドが０よりも大きい場合、４個のスカラレジスタと１６個のベクトルレジスタが、図４に示すように各々が４つの循環レジスタからなる４つのバンク形態で使用できる。ベクトルレジスタの最初のバンクＶ０－Ｖ３はスカラレジスタＤ０－Ｄ３とオーバラップする。レジスタはスカラとしてアドレスされるか、あるいは各オペランドに対して選択されたレジスタに応じてアドレスされる。詳細はセクション０、３．４レジスタ使用法を参照。



図例４　循環倍精度レジスタ

セクション０の単精度の例と同様に、倍精度レジスタは４つのバンク内で循環する。

【０１２１】

３．４　レジスタ使用法

スカラとベクトルの間で３つの演算がサポートされている。（OP_2は浮動小数点コプロセッサによってサポートされている２つのオペランド演算のいずれでもよい；OP_3は３つのオペランド演算のいずれでもよい。）

【０１２２】

以下の説明において、レジスタファイルの’最初のバンク’は単精度演算の場合レジスタＳ０－Ｓ７として、また倍精度演算の場合レジスタＤ０－Ｄ３として定義される。

・　ScalarD = OP_2 ScalarA or ScalarD = ScalarA OP_3 ScalarB or ScalarD = ScalarA * ScalarB + ScalarD

・　VectorD = OP_2 ScalarA or VectorD = ScalarA OP_3 VectorB or VectorD = ScalarA * VectorB + VectorD

・　VectorD = OP_2 VectorA or VectorD = VectorA OP_3 VectorB or VectorD = VectorA * VectorB + VectorD

【０１２３】

３．４．１　スカラ演算

２つの条件によってＦＰＳがスカラモードで動作する。

【０１２４】

１？　ＦＰＳＣＲ内のLENフィールドが０。転送先および転送元レジスタは、単精度演算の場合スカラレジスタ０-３１のいずれでもよく、また倍精度演算の場合０-１５のいずれでもよい。演算は命令ではっきりと指定されたレジスタのみに対して実行される。

【０１２５】

２？　転送先レジスタがレジスタファイルの最初のバンクにある。転送元スカラは他のレジスタのいずれでもよい。このモードはＦＰＳＣＲ内のLENフィールドを変えることなくスカラ演算とベクトル演算を混在させることを可能にする。

【０１２６】

３．４．２　ベクトル転送先を持つ、スカラとベクトル転送元を使った演算

このモードでの演算は、ＦＰＳＣＲ内のＬＥＮフィールドがゼロよりも大きく、転送先レジスタがレジスタファイルの最初のバンクにない場合に行われる。転送元スカラレジスタはレジスタファイルの最初のバンクのいずれのレジスタでもよく、残りのレジスタはいずれもＶｅｃｔｏｒＢに使用することができる。転送元スカラレジスタがＶｅｃｔｏｒＢのメンバーであるかあるいはＶｅｃｔｏｒＤがLEN要素より短い長さでＶｅｃｔｏｒＢにオーバラップしている場合、その挙動は予測不可能である。つまり、ＶｅｃｔｏｒＤとＶｅｃｔｏｒＢは同じベクトルかあるいはすべてのメンバーにおいて完全に区別されていなければならない。セクション０のサマリテーブルを参照。

【０１２７】

３．４．３　ベクトルデータのみを使った演算

このモードでの演算は、ＦＰＳＣＲ内のＬＥＮフィールドがゼロよりも大きく、転送先ベクトルレジスタがレジスタファイルの最初のバンクにない場合に行われる。ＶｅｃｔｏｒＡベクトルの個々の要素はＶｅｃｔｏｒＢにおける対応する要素と組合わせられてＶｅｃｔｏｒＤに書き込まれる。レジスタファイルの最初のバンク内にないレジスタはいずれもＶｅｃｔｏｒＡが使用でき、すべてのベクトルはＶｅｃｔｏｒＢに使用できる。２番目の場合と同様に、転送元ベクトルと転送先ベクトルのいずれかがＬＥＮ要素よりも短い長さでオーバラップする場合、その挙動は予測不可能である。それらは同一かあるいはすべてのメンバーにおいて完全に区別されていなければならない。セクション０のサマリテーブルを参照。

ＦＭＡＣファミリの演算の場合、転送先レジスタまたはベクトルは常に累算レジスタまたはベクトルである。

【０１２８】

３．４．４　演算サマリテーブル

以下の表は、単・倍精度２・３オペランド命令に対するレジスタ使用に関するオプションを示すものである。’Ａｎｙ(いずれでも)’とは指定されたオペランドに対する精度においてすべてのレジスタが使用できることを示している。

表2　単精度3オペランドレジスタ使用法

LEN フィールド	転送先 Reg	第1転送元 Reg	第2転送元 Reg	演算タイプ
0	どれでも	どれでも	どれでも	S = S op S or S = S * S + S
0でない	0-7	どれでも	どれでも	S = S op S or S = S * S + S
0でない	8-31	0-7	どれでも	V = S op V or V = S * V + V
0でない	8-31	8-31	どれでも	V = V op V or V = V * V + V

表3　単精度2オペランドレジスタ使用法

LEN フィールド	転送先 Reg	転送元 Reg	演算タイプ
0	どれでも	どれでも	S = op S
0でない	0-7	どれでも	S = op S
0でない	8-31	0-7	V = op S
0でない	8-31	8-31	V = op V

表4 倍精度3オペランドレジスタ使用法

LEN フィールド	転送先 Reg	第1転送元 Reg	第2転送元 Reg	演算タイプ
0	どれでも	どれでも	どれでも	S = S op S or S = S * S + S
0でない	0-3	どれでも	どれでも	S = S op S or S = S * S + S
0でない	4-15	0-3	どれでも	V = S op V or V = S * V + V
0でない	4-15	4-15	どれでも	V = V op V or V = V * V + V

表5 倍精度2オペランドレジスタ使用法

LEN フィールド	転送先 Reg	転送元 Reg	演算タイプ
0	どれでも	どれでも	S = op S
0でない	0-3	どれでも	S = op S
0でない	4-15	0-3	V = op S
0でない	4-15	4-15	V = op V

【０１２９】

４． 命令セット

ＦＰＳ命令は３つのカテゴリに分割できる。

・ ＭＣＲとＭＲＣ：ＡＲＭとＦＰＳとの間の転送

・ ＬＤＣとＳＴＣ：ＦＰＳとメモリとの間のロードおよびストア

・ ＣＤＰ:データ処理

【０１３０】

４．１ 命令同時性

ＦＰＳアーキテクチャ仕様の意図には２つのレベルがある。パイプライン機能ユニットと、ＣＤＰ機能を有する並行ロード／ストア動作。現在処理中の動作と並行に実行する際これらの動作とはレジスタ依存性を持たないロードとストア動作をサポートすることによって大きな性能向上が得られる。

【０１３１】

４．２　命令の逐次化

ＦＰＳでは、ＡＲＭが現在実行中のすべての命令が完了するまで、またそれぞれの例外状態が分かるまでＦＰＳを待たせる単一命令を規定している。もし例外が未決定であれば逐次化命令は中止され、例外処理はＡＲＭの中で開始される。FPSにおける逐次化命令は：

・　ＦＭＯＶＸ:浮動小数点システムレジスタに対して読込むまたは書込む

【０１３２】

浮動小数点システムレジスタに対する読込みまたは書込みは現在の命令が完了するまで中断される。System ID Register（FPSID)に対するFMOVXは先行する浮動小数点命令によって発生した例外によって開始される。User Status and Control Register（ＦＰＳＣＲ)に対して(FMOVXを使って)の読込み／変更／書込みを行って例外状態ビット（ＦＰＳＣＲ［４：０])をクリアすることができる。

【０１３３】

４．３　整数データを使う変換

浮動小数点と整数データとの間の変換は、ＦＰＳにおいて２ステッププロセスである。すなわち整数データを扱うデータ転送命令と変換を行うＣＤＰ命令からなる。整数フォーマットのままでＦＰＳレジスタの整数データに対して算術演算を試みた場合、結果は予測不可能であり、そのような演算は避けなければならない。

【０１３４】

４．３．１　ＦＰＳレジスタの整数データから浮動小数点データへの変換

整数データは、ＭＣＲ　ＦＭＯＶＳ命令を使ってＡＲＭレジスタから浮動小数点単精度レジスタへロードすることができる。そしてＦＰＳレジスタ内の整数データは、一連の整数／浮動小数点変換演算によって単精度または倍精度浮動小数

点値に変換されて転送先ＦＰＳレジスタに書き込まれる。整数値が必要ない場合、転送先レジスタは転送元レジスタであってもよい。整数は符号付または符号なし３２ビット量とすることができる。

【０１３５】

４．３．２　ＦＰＳレジスタの浮動小数点データから整数データへの変換

ＦＰＳ単精度または倍精度レジスタの値は、一連の浮動小数点／整数変換演算によって符号付または符号なし３２ビット整数フォーマットに変換できる。得られた整数は転送先単精度レジスタに入れられる。整数データはＭＲＣ　ＦＭＯＶＳ命令を使ってＡＲＭレジスタに格納することができる。

【０１３６】

４．４　レジスタファイルのアドレスによるアクセス

単精度スペース（Ｓ＝０）内で動作する命令はオペランドアクセスのために命令フィールド内の５ビットを使う。上位４ビットはＦｎ、ＦｍまたはＦｄとラベルされたオペランドフィールドに含まれる。アドレスの最下位ビットはＮ、Ｍ、またはＤに入っている。

【０１３７】

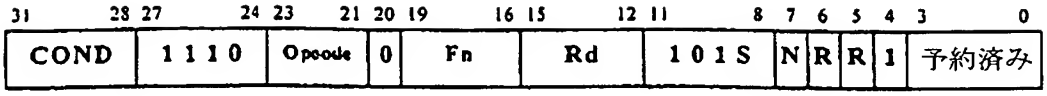
倍精度スペース（Ｓ＝１）内で動作する命令はオペランドアドレスの上位４ビットのみを使用する。これら４ビットはＦｎ、ＦｍおよびＦｄフィールドに含まれる。Ｎ、Ｍ、およびＤビットは、対応するオペランドフィールドにオペランドアドレスが入っているときは０でなければならない。

【０１３８】

４．５　ＭＣＲ（ＡＲＭレジスタからコプロセッサへ移動）

ＭＣＲ動作はＡＲＭレジスタ内のデータをＦＰＳによって転送または使用することである。これは、単精度フォーマットにおいてはＡＲＭレジスタから、また倍精度フォーマットにおいては一対のＡＲＭレジスタからＦＰＳレジスタへデータを移動させ、符号付または符号なし整数値をＡＲＭレジスタから単精度ＦＰＳレジスタへロードし、さらにＡＲＭレジスタの内容を制御レジスタにロードする動作を含む。

ＭＣＲ命令のフォーマットは図例５に示す。



図例5 MCR命令フォーマット

表6 MCRビットフィールドの定義

ビットフィールド	定義
Opcode	3ビット演算コード(表7参照)
Rd	ARM転送元レジスタコード
S	演算オペランドサイズ 0： 単精度オペランド 1： 倍精度オペランド
N	単精度演算： 転送先レジスタの最下位ビット 倍精度演算： 0に設定しなければならない。さもなければ演算は未定義。 システムレジスタは以下を移動する 予約済み
Fn	単精度演算： 転送先レジスタアドレス上位4ビット 倍精度演算： 転送先レジスタアドレス システムレジスタは以下を移動する： 0000-FPID（コプロセッサID番号） 0001-FPSCR(ユーザステータスおよび制御レジスタ) 0100-FPREG（レジスタファイル内容レジスタ) 他のレジスタのコードは予約済みであり、インプリメンテーションによって異なることがある。
R	予約ビット

表７　ＭＣＲ演算コードフィールド定義

Opcode フィールド	名称	動作
０００	ＦＭＯＶＳ	F=Rd(３２ビット、コプロセッサ１０)
０００	ＦＭＯＶＬＤ	Low(Fn)=Rd(倍精度下位３２ビット、コプロセッサ１１)
００１	ＦＭＯＶＨＤ	High(Fn)=Rd(倍精度上位３２ビット、コプロセッサ１１)
０１０－１１０	予約済み	
１１１	ＦＭＯＶＸ	System Reg=Rd (コプロセッサ１０スペース)

【０１３９】

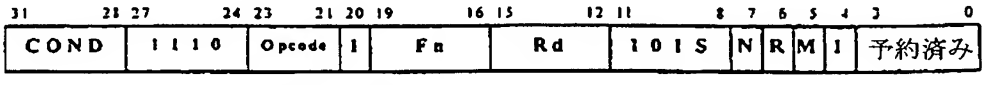
注：３２ビットデータ処理のみＦＭＯＶ［Ｓ，ＨＤ，ＬＤ］命令によってサポートされている。ＡＲＭレジスタまたは単精度レジスタのデータのみがＦＭＯＶＳ動作によって移動される。２つのＡＲＭレジスタから倍精度オペランドを転送する際、ＦＭＯＶＬＤとＦＭＯＶＨＤ命令がそれぞれ下半分と上半分を移動する。

【０１４０】

４．６　ＭＲＣ（コプロセッサ／比較浮動小数点レジスタからＡＲＭレジスタへ移動）

ＭＲＣ動作はＦＰＳレジスタのデータをＡＲＭレジスタへ転送する。これは、単精度値、または浮動小数点値を整数に変換した結果、をＡＲＭレジスタへ移動、あるいは倍精度ＦＰＳレジスタを２つのＡＲＭレジスタへ移動し、ＣＰＳＲのステータスビットを前の浮動小数点比較演算の結果で変更する動作を含む。

ＭＲＣ命令のフォーマットを図例６に示す



図例6　MRC命令フォーマット

表８　ＭＲＣビットフィールド定義

ビットフィールド	定義
Opcode	3ビットＦＰＳ演算コード(表９参照)
Rd	ＡＲＭ転送元＊レジスタコード
S	演算オペランドサイズ ０：　単精度オペランド １：　倍精度オペランド
N	単精度演算： 転送先レジスタの最下位ビット 倍精度演算： ０に設定しなければならない。さもなければ演算は未定義。 システムレジスタは以下を移動する 予約済み
M	予約済み
Fn	単精度演算： 転送先レジスタアドレス上位4ビット 倍精度演算： 転送先レジスタアドレス システムレジスタは以下を移動する： ００００-ＦＰＩＤ（コプロセッサＩＤ番号） ０００１-ＦＰＳＣＲ(ユーザステータスおよび制御レジスタ) ０１００-ＦＰＲＥＧ（レジスタファイル内容レジスタ) 他のレジスタのコードは予約済みであり、インプリメンテーションによって異なることがある。
Fm	予約ビット
R	予約ビット

＊ＦＭＯＶＸ　ＦＰＳＣＲ命令の場合、もしＲｄフィールドにＲ１５(１１１１)が入っているなら、ＣＰＳＲの上位４ビットは得られた条件コードで更新される。

表９　ＭＲＣ演算コードフィールド定義

Opcode フィールド	名称	動作
０００	ＦＭＯＶＳ	Rd=Fn（３２ビット、コプロセッサ１０）
０００	ＦＭＯＶＬＤ	Rd=Low(Fn)Dnの下位32ビッドが転送される。（倍精度下位３２ビット、コプロセッサ１１）
００１	ＦＭＯＶＨＤ	Rd=High(Fn)Dnの上位３２ビットが転送される。（倍精度上位３２ビット、コプロセッサ１１）
０１０－１１０	予約済み	
１１１	FMOVX	Rd=System Reg

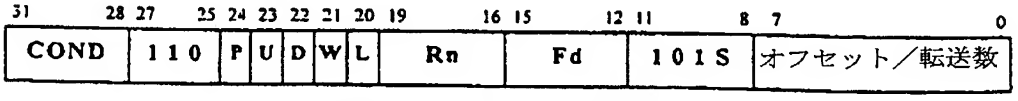
注：ＭＣＲ　ＦＭＯＶ命令の注記を参照。

【０１４１】

４．７　ＬＣＤ／ＳＴＣ（ロード／ストアＦＰＳレジスタ）

ＬＤＣとＳＴＣ動作はＦＰＳとメモリとの間でデータを転送する。浮動小数点データは、ＡＲＭアドレスレジスタを更新してあるいはそのままの状態で、単一データ転送または複数のデータ転送によっていずれの精度でも転送できる。満杯の降順スタックおよび空の昇順スタックの両方の構造がサポートされている。さらに複数移動命令によってデータ構造に対する複数オペランドアクセスもサポートされている。ＬＤＣとＳＴＣの各種オプションの説明については表１１を参照。

ＬＤＣとＳＴＣ命令のフォーマットを図例７に示す。



図例7　LDC/STC命令フォーマット

表１０　ＬＤＣ／ＳＴＣビットフィールド定義

ビットフィールド	定義
P	前／後インデクシング（０＝後、１=前）
U	上／下ビット（０＝下、１＝上）
D	単精度演算： 転送元/転送先レジスタ最下位ビット 倍精度演算： ０に設定しなければならない
W	ライトバックビット（０=ライトバックなし、１=ライトバック）
L	方向ビット（０=ストア、１=ロード）
Rn	ARMベースレジスタコード
Fd	単精度演算： 転送元/転送先レジスタアドレスでアクセス上位４ビット 倍精度演算： 転送元/転送先レジスタアドレス
S	演算オペランドサイズ ０：単精度オペランド １：倍精度オペランド
ｵﾌｾｯﾄ/転送数	FLDM(IA/DB)およびFSTM(IA/DB)に対して転送すべき符号なし８ビットオフセットまたは単精度レジスタ数（倍精度レジスタ数の２倍）。１回の転送の最大のワード数は１６。これで１６個の単精度値または８個の倍精度値の転送が可能。

【０１４２】

４．７．１　ロードとストア動作に関する一般的注意点

複数レジスタのロードおよびストアは、ベクトル演算によって使われる４または８個のレジスタからなる範囲での回り込みをしないで、直線的にレジスタファイルを介して行う。レジスタファイルの終端を越えてロードを試みた場合、結果は予測不可能である。

【０１４３】

ダブルロードまたは複数ストアのオフセットに１７またはそれ以下の奇数のレジスタ数が入っている場合、インプリメンテーションではさらに別の３２ビットデータ項目を書き込むかまたは別の３２ビットデータ項目を読むことができるが、必ずしもそのようにする必要はない。この追加データ項目を使ってレジスタが

ロードまたはストアされる際にそれらの内容を特定することができる。これは、レジスタファイルフォーマットがその精度のＩＥＥＥ７５４フォーマットとは異なり、各レジスタがそれ自身のメモリ内識別のために必要なタイプ情報を有する場合に、役に立つ。オフセットが奇数で単精度レジスタの数よりも大きい場合、これはレジスタのコンテクスト切替とすべてのシステムレジスタを始動させるのに使うことができる。

表１１　ロードおよびストアアドレス指定モードオプション

P	W	オフセット／転送数	アドレス指定モード	名称
タイプ０転送：ライトバックなしに複数をロード/ストアする				
0	0	転送すべきレジスタの数	FLDM<cond><S/D>Rn.<レジスタリスト> FSTM<cond><S/D>Rn.<レジスタリスト>	ロード/ストアマルチプル
Rnにおける先頭アドレスから複数のレジスタをロード/ストア。Rnの変更なし。レジスタの数は単精度の場合１-１６個、倍精度の場合１-８個。オフセットフィールドには３２ビット転送の数が入っている。このモードを使ってグラフィックス演算用の変換マトリックスおよびその変換点をロードすることができる。				
例： FLDMEQSr12,{f8-f11}；ｒ１２のアドレスから４個の単精度を４個のｆｐレジスタｓ８、ｓ９、ｓ１０へロードする。ｒ１２は不変。 FSTMEQDr4,{f0}；ｄ０から１個の倍精度をｒ４のアドレスへストアする。ｒ４は不変。				
タイプ１転送：複数をロード/ストアする。Rnのポストインデックスとライトバックあり。				
0	1	転送すべきレジスタの数	FLDM<cond>IA<S/D>Rn!.<レジスタリスト> FSTM<cond>IA<S/D>Rn!.<レジスタリスト>	ロード/ストアマルチプル
Rnにおける先頭アドレスから複数のレジスタをロード/ストアし、最後の転送の後に次のアドレスをRnへライトバック。オフセットフィールドには３２ビット転送の数が入っている。Rnへのライトバックはオフセット＊４。ロードマルチプルにおいて転送された最大ワード数は１６。Uビットは１にセットしなければならない。これは空の昇順スタックへ格納するために、あるいは満杯の降順スタックからロードするために使用する。あるいは変換点を格納し、次の点へポインタをインクリメントするために、またフィルタ動作において複数データをロード/ストアするために使用する。				
例： FLDMEQIASr13!,{f12-f15}；ｒ１３のアドレスカラ４個の単精度を４個のｆｐレジスタｓ１２、ｓ１３、ｓ１４、ｓ１５へロードする。ｒ１３は次のデータを指すアドレスで更新する。				
タイプ２転送：１個のレジスタをロード/ストアする。Rnのプリインデックスあり、ライトバックなし。				

1	0	オフセット	FLD<cond><S/D>[Rn.#+/-offset], Fd FST<cond><S/D>[Rn.#+/-offset], Fd	オフセットありロード/ストア
１個のレジスタをロード/ストアする。Rnのアドレスのプリインクリメントあり、ライトバックなし。オフセット値はオフセット＊４であり、Rnに加える（U=1）かそれから差し引く（U=0）ことによりアドレスを発生する。これは、オペランドアクセスのために有用であり、浮動小数点データを取り出すためにメモリをアクセスする典型的な方法である。				
例： FSTEQDf4,[r8,#+8]； ３２（８＊４）バイトオフセットしたｒ８のアドレスカラ１個の倍精度をｄ４へ格納する。ｒ８は不変。				
タイプ３転送：複数レジスタをロード/ストアする。プリインデックスとライトバックあり。				
1	1	転送すべきレジスタの数	FLDM<cond>DB<S/D>Rn!.<レジスタリスト> FSTM<cond>DB<S/D>Rn!.<レジスタリスト>	プリデクリメント有りロード/ストアマルチプル
複数のレジスタをロード/ストアする。Rnのアドレスをプリデクリメントし、新しいターゲットアドレスをRnにライトバック。オフセットフィールドには３２ビット転送の数が入っている。ライトバック値はオフセット＊４であり、Rnから差し引く（U=0）。このモードは満杯の降順スタックへストアするか、空の昇順スタックからロードするのに使用する。				
例： FSTMEQDBSr9!,{f27-f29}； ｓ２７、ｓ２８、ｓ２９から３個の単精度を満杯の降順スタックへストアする。最後のエントリアドレスはｒ９に入っている。ｒ９は新しい最後のエントリを指すように更新する。				

【０１４４】

４．７．２　ＬＤＣ／ＳＴＣ動作サマリ

表１２はＬＤＣ／ＳＴＣ演算コードにおけるＰ、Ｗ、Ｕビットの許容される組合わせと、各有効な演算に対するオフセットの機能を示す。

表１２　ＬＤＣ／ＳＴＣ動作サマリ

P	W	U	オフセットフィールド	動作
0	0	0		未定義
0	0	1	レジスタカウント	FLDM/FSTM
0	1	0		未定義
0	1	1	レジスタカウント	FLDM1A/FSTM1A
1	0	0	オフセット	FLD/FST
1	0	1	オフセット	FLD/FST
1	1	0	レジスタカウント	FLDMDB/FSTMDB
1	1	1		未定義

【０１４５】

４．８　ＣＤＰ(コプロセッサデータ処理)

ＣＤＰ命令は、浮動小数点レジスタファイルからのオペランドを使用して結果を生み出し、その結果がレジスタファイルに書き戻されるようなすべてのデータ処理動作を含む。特に興味があるのはＦＭＡＣ(乗算-累算がつながった)演算であり、これは２つのオペランドを掛け合わせ、３番目のオペランドを加える演算である。この演算は、ＩＥＥＥ丸め演算が３番目のオペランドを加える前に積に対して行われる点で、融合乗累算演算とは異なる。これにより、Ｊａｖａコードは ＦＭＡＣ演算を利用することにより、乗算の後に加算する演算よりも乗累算演算の速度を速めることができる。

【０１４６】

ＣＤＰグループ内の２つの命令はＦＰＳレジスタ内の浮動小数点値をその整数値に変換する上で役に立つ。ＦＦＴＯＵＩ［Ｓ／Ｄ]は、ＦＰＳＣＲ内の現行丸めモードを使って、単精度または倍精度レジスタの内容をＦＰＳレジスタ内の符号なし整数へ変換する。ＦＦＴＯＳＩ［Ｓ／Ｄ］は符号付整数への変換を行う。ＦＦＴＯＵＩＺ［Ｓ／Ｄ］とＦＦＴＯＳＩＺ［Ｓ／Ｄ］は同じ機能を行うが、変換用ＦＰＳＣＲ丸めモードを無効にして、小数ビットを切り捨てる。ＦＦＴＯＳＩＺ［Ｓ／Ｄ]の機能は浮動小数点から整数に変換する際にＣ、Ｃ＋＋およびＪ

ａｖａで要求される。ＦＦＴＯＳＩＺ［Ｓ／Ｄ］命令はＦＰＳＣＲからＲＺまでの変換用丸めモードビットを調整することなくこの能力を提供するので、変換のためのサイクルカウントをＦＦＴＯＳＩＺ［Ｓ／Ｄ］演算のサイクルカウントにまで減少し、４－６サイクル節約する。

【０１４７】

比較演算はＣＤＰ　ＣＭＰ命令、それに続くＭＲＣ　ＦＭＯＶＸ　ＦＰＳＣＲ命令を使って行い、得られたＦＰＳフラグビット（ＦＰＳＣＲ[31:28]）をＡＲＭＣＰＳＲフラグビットにロードする。比較命令は、比較オペランドの１つがＮａＮである場合、ＩＮＶＡＬＩＤ（無効）例外の可能性があることもないこともある。ＦＣＭＰとＦＣＭＰ0は、比較オペランドの１つがＮａＮである場合、ＩＮＶＡＬＩＤ（無効）を知らせないが、ＦＣＭＰＥとＦＣＭＰＥ０は例外を知らせる。ＦＣＭＰ０とＦＣＭＰＥ０はＦｍフィールド内のオペランドと０とを比較し、その結果にしたがってＦＰＳフラグをセットする。ＡＲＭフラグＮ、Ｚ、Ｃ、ＶはＦＭＯＶＸ　ＦＰＳＣＲ命令の後に以下のように定義される。

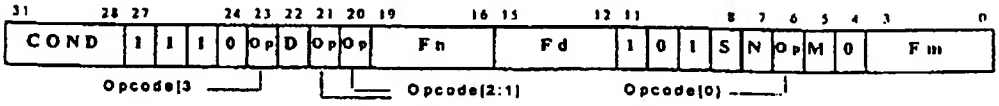
Ｎ：　よりも少ない

Ｚ：　等しい

Ｃ：　よりも大きい、または等しい、または順序なし

Ｖ：　順序なし

ＣＤＰ命令のフォーマットは図例８に示す。



図例8　CDP命令フォーマット

表１３　ＣＤＰビットフィールド定義

ビットフィールド	定義
Opcode	４ビットＦＰＳ演算コード（表１４参照）
D	単精度演算： 転送先レジスタ最下位ビット 倍精度演算： ０にセットしなければならない
Fn	単精度演算： 転送元Aレジスタ４ビットまたは 演算コード最上位４ビットを拡張 倍精度演算： 転送元Aレジスタアドレスまたは 演算コード最上位４ビットを拡張
Fd	単精度演算： 転送先レジスタ４ビット 倍精度演算： 転送先レジスタアドレス
S	演算オペランドサイズ ０：単精度オペランド １：倍精度オペランド
N	単精度演算： 転送元Aレジスタ最下位ビット 演算コード最下位ビットを拡張 倍精度演算： ０にセットしなければならない 演算コード最下位ビットを拡張
M	単精度演算： 転送元Bレジスタ最下位ビット 倍精度演算： ０にセットしなければならない
Fm	単精度演算： 転送元Bレジスタアドレス上位４ビット 倍精度演算： 転送元Bレジスタアドレス

【０１４８】

４．８．１　演算コード

表１４はＣＤＰ命令の基本演算コードを示す。すべてのニーモニックコードは[OPERATION][COND][S/D]という形を取る。

表１４　ＣＤＰ演算コード仕様

演算コードフィールド	命令名称	演算
００００	FMAC	Fd=Fn*Fm+Fd
０００１	FNMAC	Fd=-(Fn*Fm+Fd)
００１０	FMSC	Fd=Fn*Fm-Fd
００１１	FNMSC	Fd=-(Fn*Fm-Fd)
０１００	FMUL	Fd=Fn*Fm
０１０１	FNMUL	Fd=-(Fn*Fm)
０１１０	FSUB	Fd=Fn-Fm
０１１１	FNSUB	Fd=-(Fn-Fm)
１０００	FADD	Fd=Fn+Fm
１００１-１０１１	予約済み	
１１００	FDIV	Fd=Fn/Fm
１１０１	FRDIV	Fd=Fm/Fn
１１１０	FRMD	Fd=Fn%Fm(Fd=Fn/Fmの後に残る小数)
１１１１	拡張	Fnレジスタフィールドを使って２オペランド演算のための命令を指定する(表１５を参照)

【０１４９】

４．８．２　拡張演算

表１５は演算コードフィールドのExtend(拡張値)を使う拡張命令を示す。すべての命令は[OPERATION][COND][S/D]の形を取るが、直列化およびＦＬＳＣＢ命令は例外である。拡張演算の命令コードはＦｎオペランドのレジスタファイルへのインデックスすなわち｛Ｆｎ［３：０］，Ｎ｝と同じようにして作られる。

表１５　ＣＤＰ拡張命令

Fn｜N	名称	演算
００００００	FCPY	Fd=Fm
００００１	FABS	Fd=abs(Fm)
００００１０	FNEG	Fd=-(Fm)
００００１１	FSQRT	Fd=sqrt(Fm)
００１００-００１１１	予約済み	
０１０００	FCMP*	Flags:Fd⇔Fm
０１００１	FCMPE*	Flags:=Fd⇔Fm例外報告有り
０１０１０	FCMP0*	Flags:=Fd⇔0
０１０１１	FCMPE0*	Flags:=Fd⇔0例外報告有り
０１１００-０１１１０	予約済み	
０１１１１	FCVTD<cond>S*	Fd(倍精度レジスタコード)=Fm(単精度レジスタコード)単精度から倍精度に変換されたもの。（コプロセッサ１０）
０１１１１	FCVTS<cond>D*	Fd(単精度レジスタコード)=Fm(倍精度レジスタコード)倍精度から単精度に変換されたもの。（コプロセッサ１１）
１００００	FUITO*	Fd=符号なし整数を単/倍精度に変換（Fm)
１００００１	FSITO*	Fd=符号あり整数を単/倍精度に変換（Fm)
１００１０-１０１１１	予約済み	
１１０００	FFTOUI*	Fd=符号なし整数に変換（Fm)｛現行RMODE｝
１１００１	FFTOSIZ*	Fd=符号なし整数に変換（Fm)｛RZモード｝
１１０１０	FFTOSI*	Fd=符号あり整数に変換（Fm)｛現行RMODE｝
１１０１１	FFTOSIZ*	Fd=符号あり整数に変換（Fm)｛RZモード｝
１１１００-１１１１１	予約済み	

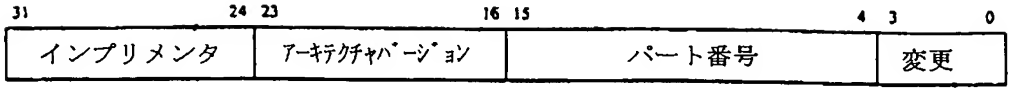
＊ベクトル化不可能な命令。LENフィールドは無視され、スカラ演算が指定されたレジスタに対して行われる。

【０１５０】

５．　システムレジスタ

５．１　システムＩＤレジスタ（ＦＰＳＩＤ）

ＦＰＳＩＤにはＦＰＳアーキテクチャおよびインプリメンテーション定義ＩＤ値が含まれる。



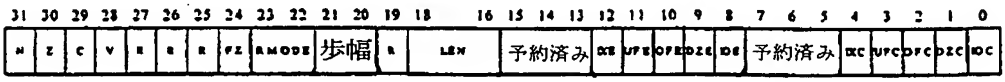
図例9　FPSIDレジスタコード

このワードを使ってＦＰＳとマスクセット番号のモデル、特徴セットおよび変更を決めることができる。ＦＰＳＩＤは読込み専用であり、ＦＰＳＩＤへの書き込みは無視される。ＦＰＳＩＤレジスタレイアウトについては図例９を参照。

【０１５１】

５．２　ユーザステータスと制御レジスタ（ＦＰＳＣＲ）

ＦＰＳＣＲレジスタはユーザがアクセス可能なコンフィギュレーションビットと例外ステータスビットとを含む。コンフィギュレーションオプションには例外許可ビット、丸め制御、ベクトル歩幅および長さ、非正規オペランドと結果の取扱い、およびデバッグモードの使用が含まれる。このレジスタはユーザとオペレーティングシステムが使用するもので、ＦＰＳを構成したり、完了した命令の状態を問い合わせるのに用いる。これはセーブしてコンテクスト切替時に回復しなければならない。ビット３１から２８は最も新しい比較命令からのフラグ値を有し、ＦＰＳＣＲの読込みを使ってアクセスできる。ＦＰＳＣＲは図例１０に示す。

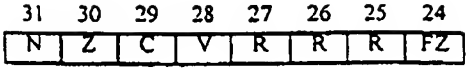


図例10　ユーザステータスと制御レジスタ(FPSCR)

【０１５２】

５．２．１　ステータス比較および処理制御バイト

ビット３１から２８は最も新しい比較命令の結果および特殊状況においてFPSの算術応答を指定するのに有用ないくつかの制御ビットを有する。ステータス比較および処理制御バイトのフォーマットは図例１１に示す。



図例11　FPSCRステータス比較および処理制御バイト

表１６　ＦＰＳＣＲステータス比較および処理制御バイトフィールド定義

レジスタビット	名称	機能
３１	N	比較結果は…より小さい
３０	Z	比較結果は…と等しい
２９	C	比較結果は…より大きいまたは等しいまたは順列なし
２８	V	比較結果は順列なし
２７：２５	予約済み	
２４	FZ	ゼロにクリア ０：IEEE７５４アンダフロー処理（デフォルト） １：小さい値の結果はゼロにする 転送先精度の正規範囲よりも小さい結果は、転送先レジスタにゼロを書き込む。アンダフロー例外トラップは取らない。

【０１５３】

５．２．２　システム制御バイト

システム制御バイトは丸めモード、ベクトル歩幅およびベクトル長さフィールドを制御する。ビットは図例１２に示すように指定される。

【０１５４】

ＶＦＰｖ１アーキテクチャはベクトル演算に使用するレジスタファイル歩幅機構を組み込んでいる。歩幅ビットが００にセットされている場合、ベクトル演算において次に選択されるレジスタはレジスタファイル内で前のレジスタの直後のレジスタとなる。正規レジスタファイル回り込み機構は歩幅値によっては影響されない。歩幅値が１１の場合、すべての入力レジスタと出力レジスタを２だけインクリメントする。

例えば

FMULEQS F8, F16, F24

は次の非ベクトル演算を実行する：

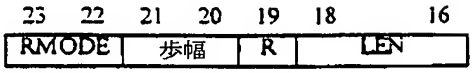
FMULEQS F8, F16, F24

FMULEQS F10, F18, F26

FMULEQS F12, F20, F28

FMULEQS F14, F22, F30

実質的にレジスタファイル内の乗算用オペランドを１レジスタではなく２レジスタずつ跨いでいる。



図例12 FPSCRシステム制御バイト

表１７　ＦＰＳＣＲシステム制御バイトフィールド定義

レジスタビット	名称	機能
２３：２２	RMODE	丸めモードを設定 ００：RN(四捨五入；デフォルト) ０１：RP(正の無限大に丸める) １０：RM（負の無限大に丸める） １１：RZ(ゼロに丸める)
２１：２０	歩幅	ベクトルレジスタアクセスを以下に設定： ００：１(デフォルト) ０１：予約済み １０：予約済み １１：２
１９	予約済み(R)	
１８：１６	LEN(長さ)	ベクトル長さ。ベクトル演算の長さを指定する。（すべてのコードがそれぞれのインプリメンテーションで使用できるのではない) ０００：１(デフォルト) ００１：２ ０１０：３ ０１１：４ １００：５ １０１：６ １１０：７ １１１：８

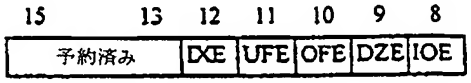
【０１５５】

５．２．３　例外許可バイト

例外許可バイトはビット１５：８を占有し、例外トラップ用イネーブルを有する。ビットは図例１３に示すように指定される。例外イネーブルビットは、浮動小数点例外条件の処理のためのＩＥＥＥ７５４使用の要求に合致している。そのビットがセットされると例外が許可され、ＦＰＳは、現在の命令に関して例外条件が発生した場合オペレーティングシステムへユーザ可視トラップを知らせる。そのビットがクリアされると例外は許可されず、ＦＰＳは例外が発生してもオペレーティングシステムに対してユーザ可視トラップを知らせない。しかし数学的に合理的な結果を発生させる。例外イネーブルビットのデフォルトは禁止される。例外処理についての詳細についてはＩＥＥＥ７５４標準を参照のこと。

【０１５６】

インプリメンテーションによっては、例外が使用禁止になっていても、ハードウェアの能力外の例外条件を取り扱うためにサポートコードへのバウンスを発生させることがある。これは一般にユーザコードには見える。



図例13 FPSCR例外イネーブルバイト

表１８　ＦＰＳＣＲ例外イネーブルバイトフィールド

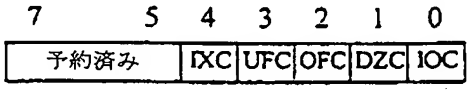
レジスタビット	名称	機能
１５：１３	予約済み	
１２	IXE	不正確なイネーブルビット ０：禁止(デフォルト) １：許可
１１	UFE	アンダフローイネーブルビット ０：禁止(デフォルト) １：許可
１０	OFE	オーバフローイネーブルビット ０：禁止(デフォルト) １：許可
９	DZE	ゼロで割るイネーブルビット ０：禁止(デフォルト) １：許可
８	IOE	無効オペランドイネーブルビット ０：禁止(デフォルト) １：許可

【０１５７】

５．２．４　例外ステータスバイト

例外ステータスバイトはＦＰＳＣＲのビット７：０を占有し、例外ステータスフラグビットを有する。５つの例外ステータスフラグビットがあり、各浮動小数点例外に１フラグビットずつ対応している。これらのビットは’接着’しており、一旦検出された例外によってセットされたならば、ＦＰＳＣＲへ書き込むＦＭＯＶＸ命令またはＦＳＥＲＩＡＬＣＬ命令によってクリアしなければならない。

これらビットは図例１４に示すように指定される。例外が許可された場合、対応する例外ステータスビットは自動的にはセットされない。必要に応じて適切な例外ステータスビットをセットするのはサポートコードの役割である。ある例外は自動的に行ってもよい、すなわち例外条件が検出されたならば、ＦＰＳは、例外イネーブルビットがどのようにセットされたかに関わらず、後続の浮動小数点命令でバウンスする。これによってＩＥＥＥ７５４標準で要求されるより複雑な例外処理をハードウェアではなくソフトウェアで実行することができる。例として、ＦＺビットが０にセットされたアンダフロー条件がある。この場合、正しい結果は結果の指数および丸めモードによっては、非正規数であるかもしれない。ＦＰＳによって、作成者はバウンスオプションを含めた応答を選択し、サポートコードを利用して正しい結果を作りこの値を転送先レジスタに書き込むことができる。もしアンダフロー例外イネーブルビットがセットされていれば、ユーザトラップハンドラはサポートコードが命令を完了した後に呼び出される。このコードはＦＰＳの状態を変えて戻るか、あるいは処理を終了することができる。



図例14 FPSCR例外ステータスバイト

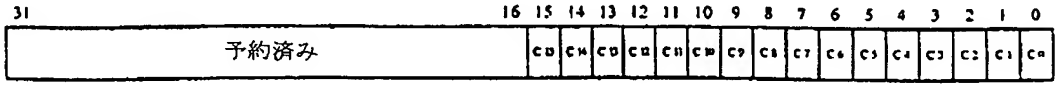
表１９　ＦＰＳＣＲ例外ステータスバイトフィールド定義

レジスタビット	名称	機能
７：５	予約済み	
4	IXC	不正確な例外検出
3	UFC	アンダフロー例外検出
2	OFC	オーバフロー例外検出
1	DZC	ゼロで割る例外検出
0	IOC	無効演算例外検出

【０１５８】

５．３　レジスタファイル内容レジスタ（ＦＰＲＥＧ）

レジスタファイル内容レジスタは、現在走っているプログラムによって解読されるままのレジスタ内容を適切に表すためにデバッガが使うことができる情報を有する特権レジスタである。ＦＰＲＥＧは１６ビットを有し、レジスタファイル内の各倍精度レジスタに対して１ビットずつ割り当てられる。ビットがセットされると、そのビットで表される物理レジスタ対が倍精度レジスタとして表示される。ビットがクリアされれば、物理レジスタは初期化されず、あるいは１つまたは２つの単精度データ値を有する。



図例15 FPREGビットフィールドの定義

表２０ ＦＰＲＥＧビットフィールド定義

ＦＰＲＥＧビット	ビットセット	ビットクリア
C0	D0有効	S1とS0有効または未初期化
C1	D1有効	S3とS2有効または未初期化
C2	D2有効	Ｓ５とＳ４有効または未初期化
C3	D3有効	Ｓ７とＳ６有効または未初期化
C4	D4有効	Ｓ９とＳ８有効または未初期化
C5	D5有効	Ｓ１１とＳ１０有効または未初期化
C6	D6有効	Ｓ１３とＳ１２有効または未初期化
C7	D7有効	Ｓ１５とＳ１４有効または未初期化
C8	D8有効	Ｓ１７とＳ１６有効または未初期化
C9	D9有効	Ｓ１９とＳ１８有効または未初期化
C10	D10有効	Ｓ２１とＳ２０有効または未初期化
C11	D11有効	Ｓ２３とＳ２２有効または未初期化
C12	D12有効	Ｓ２５とＳ２４有効または未初期化
C13	D13有効	Ｓ２７とＳ２６有効または未初期化
C14	D14有効	Ｓ２９とＳ２８有効または未初期化
C15	D15有効	Ｓ３１とＳ３０有効または未初期化

【０１５９】

６． 例外処理

ＦＰＳは２つのモード、デバッグモードと正常モード、の内いずれかのモード

で動作する。ＤＭビットがＦＰＳＣＲにセットされると、ＦＰＳはデバッグモードで動作する。このモードにおいて、ＦＰＳは一度に１つの命令を実行し、その間ＡＲＭは命令の例外状態が分かるまでまたされる。これによってレジスタファイルとメモリは命令の流れに関しては正確になるが、実行時間は大幅に増大するという犠牲を払うことになる。ＦＰＳはリソースが許すならばＡＲＭから新しい命令を受け付け、また例外条件を検出したときに例外を知らせる。ＡＲＭへの例外報告は浮動小数点命令列に関しては常に正確である。ただベクトル演算に続いてベクトル演算と並列に実行するロードまたはストア命令の場合は例外である。この場合、ロード命令に関してはレジスタファイルの内容が、またストア命令に関してはメモリの内容が正確ではなくなるかもしれない。

【０１６０】

６．１　サポートコード

ＦＰＳのインプリメンテーションはハードウェアとソフトウェアサポートを利用してＩＥＥＥ７５４に準拠するようにすることもできる。サポートされていないデータタイプや自動例外に関しては、サポートコードは準拠したハードウェアの機能を実行し結果を転送先レジスタに返してユーザのコードに戻る。その際ユーザのトラップハンドラを呼び出したりユーザのコードの流れを変更することはない。ユーザにとっては、ハードウェアのみが浮動小数点コードの処理を行ったように見える。これらの処理を取り扱うためにサポートコードにバウンスすることはこれらの処理を実行する時間を大幅に削減するが、これらの状況が発生することは普通ユーザコード、埋め込まれたアプリケーションおよび良く書かれた算術アプリケーションにおいては少ない。

【０１６１】

サポートコードは２つのコンポーネントを有するように意図されている。その１つはルーチンの集まりで、超越計算等ハードウェアの範囲を越えた演算を実行し、また例外を発生させる可能性のあるサポートされていない入力を使っての除算等、サポートされている機能を実行するものである。もう１つは例外ハンドラの集合で、ＩＥＥＥ７５４に準拠するように例外トラップを処理するものである。サポートコードは作成された機能を実行して、サポートされていないデータタ

イプまたはデータ表現(例えば、非正規値)の適切な取扱いをエミュレートしなければならない。ルーチンの出口においてユーザの状態を回復させるように注意を払うならば、ルーチンは中間計算においてＦＰＳを利用するように書くこともできる。

【０１６２】

６．２　例外報告と処理

正常モードでの例外は、例外条件が検出された後に発行される次の浮動小数点命令でＡＲＭに報告する。ＡＲＭプロセッサ、ＦＰＳレジスタファイルおよびメモリの状態は、例外が取られたときの違反命令に関しては正確ではないかもしれない。その命令を正しくエミュレートし、その命令から生じる例外を処理するのに十分な情報をサポートコードが得られる。

【０１６３】

インプリメンテーションによっては、サポートコードを使って、無限大、ＮａＮ、非正規データおよびゼロを含む特殊ＩＥＥＥ７５４データを有するいくつかのあるいはすべての命令を処理することもできる。そのような処理をするインプリメンテーションは、これらのデータをサポートされていないデータとして参照し、一般的にはユーザコードには見えないようにサポートコードにバウンスし、ＩＥＥＥ７５４指定結果を転送先レジスタに入れて戻る。この動作から生じる例外はいかなるものでも、ＩＥＥＥ７５４例外ルールに従う。もし対応する例外イネーブルビットがセットされているなら、この例外はユーザコードへのトラップも含むことができる。

【０１６４】

ＩＥＥＥ７５４標準は、ＦＰＳＣＲ内の例外ビットがイネーブルされている場合とイネーブルされていない場合の両方の場合に対して例外条件への対応を定義している。ＶＦＰｖ１アーキテクチャはＩＥＥＥ７５４仕様に適切に準拠するように使用されるハードウェアとソフトウェアとの間に境界を規定していない。

【０１６５】

６．２．１　サポートされていない命令とフォーマット

ＦＰＳは１０進データの命令あるいは１０進データへ/からの変換はサポート

していない。これらの命令はＩＥＥＥ７５４標準で要求されており、サポートコードによって提供されなければならない。１０進データを利用する場合、所望の機能のルーチンの集まりを必要とする。ＦＰＳは１０進データタイプを持っていないので、１０進データを使用する命令をトラップするために使うことはできない。

【０１６６】

６．２．２　ＦＰＳが使用禁止または例外となっている場合のＦＭＯＶＸの使用

スーパーバイザまたは未定義モードにおいて実行されるＦＭＯＶＸ命令は、ＦＰＳが例外状態または使用禁止のときに、（インプリメンテーションが使用禁止オプションをサポートしている場合）例外をＡＲＭに知らせることなくＦＰＳＣＲを読み書きあるいはＦＰＳＩＤまたはＦＰＲＥＧを読み込む。

【０１６７】

本発明の特定の実施例について述べたが、本発明がそれらに制限されないこと、また発明の範囲内で様々な変更や追加が行えることは明らかであろう。例えば、本発明の範囲を逸脱することなしに、以下の従属クレームの特徴を独立クレームの特徴と様々に組み合わせることができる。

【図面の簡単な説明】

【図１】

データ処理システムの概略図である。

【図２】

スカラレジスタとベクトルレジスタの両方をサポートする浮動小数点ユニットの説明図である。

【図３】

単精度演算において、与えられたレジスタがベクトルレジスタかスカラレジスタのいずれであるかをどのようにして決めるのかを示す流れ図である。

【図４】

倍精度演算において、与えられたレジスタがベクトルレジスタかスカラレジスタのいずれであるかをどのようにして決めるのかを示す流れ図である。

【図５】

レジスタバンクを分割したそれぞれのサブセット内で、単精度演算時の回り込みを示す図である。

【図６】

レジスタバンクを分割したそれぞれのサブセット内で、倍精度演算時の回り込みを示す図である。

【図７Ａ】

メインプロセッサが見るコプロセッサ命令、単精度および倍精度コプロセッサが見るコプロセッサ命令、および単精度コプロセッサが見るコプロセッサ命令をそれぞれ示す図である。

【図７Ｂ】

メインプロセッサが見るコプロセッサ命令、単精度および倍精度コプロセッサが見るコプロセッサ命令、および単精度コプロセッサが見るコプロセッサ命令をそれぞれ示す図である。

【図７Ｃ】

メインプロセッサが見るコプロセッサ命令、単精度および倍精度コプロセッサが見るコプロセッサ命令、および単精度コプロセッサが見るコプロセッサ命令をそれぞれ示す図である。

【図８】

単精度および倍精度コプロセッサを制御するメインプロセッサを示す図である。

【図９】

単精度コプロセッサを制御するメインプロセッサを示す図である。

【図１０】

コプロセッサ命令が受領されたことを示すためにメインプロセッサに受付信号を返すべきかどうかを決定する、単精度および倍精度コプロセッサ内の回路を示す図である。

【図１１】

コプロセッサ命令が受領されたことを示すためにメインプロセッサに受付信号を返すべきかどうかを決定する、単精度コプロセッサ内の回路を示す図である。

【図１２】

メインプロセッサ内での未定義の命令例外の取り扱いを示す図である。

【図１３】

本発明の好ましい実施例によるコプロセッサの要素を示すブロック図である。

【図１４】

本発明の好ましい実施例によるレジスタ制御および命令発行ロジックの動作を示す流れ図である。

【図１５】

本発明の好ましい実施例による浮動小数点レジスタの内容の一例である。

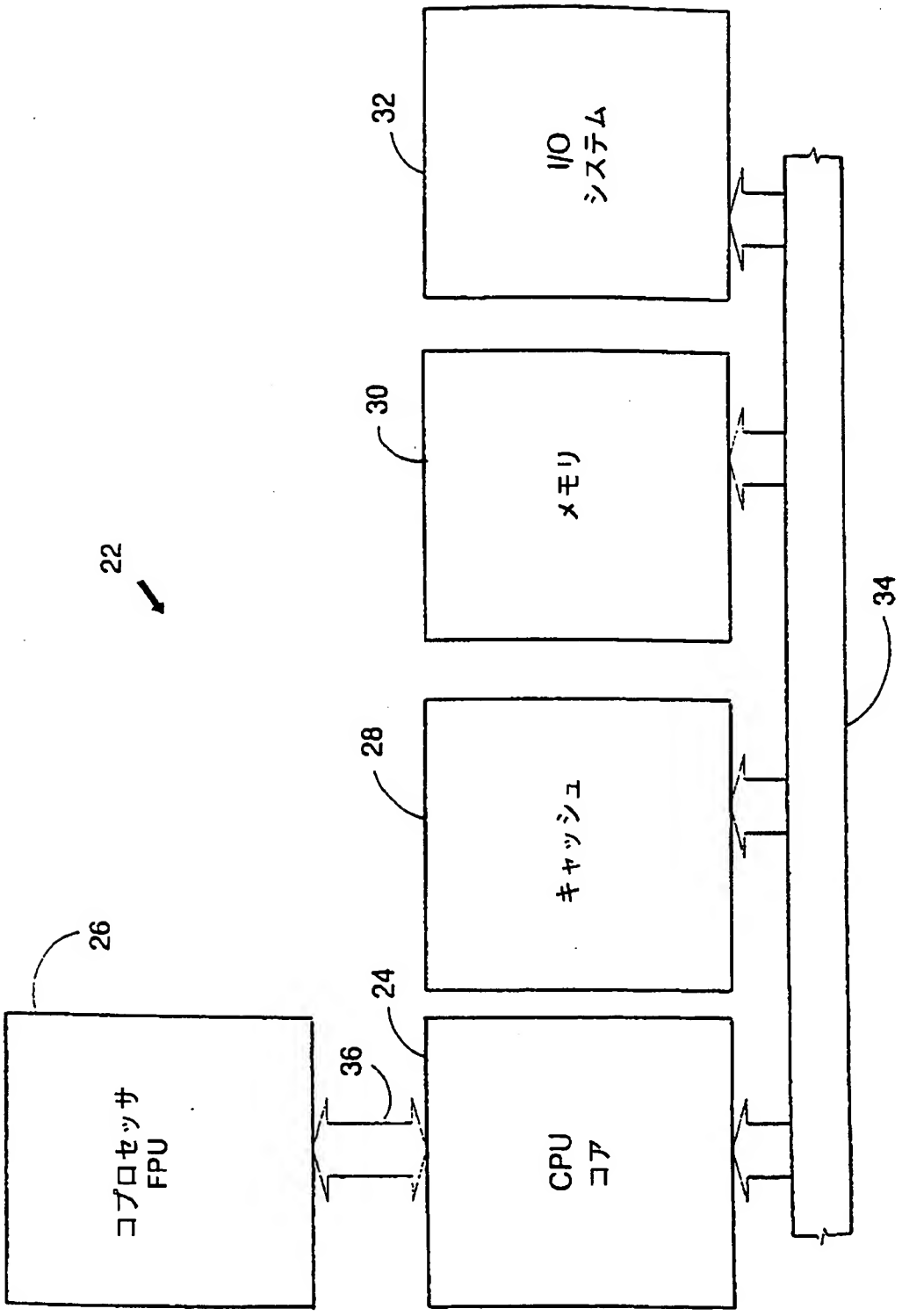
【図１６】

クレイ１プロセッサ内のレジスタバンクを示す図である。

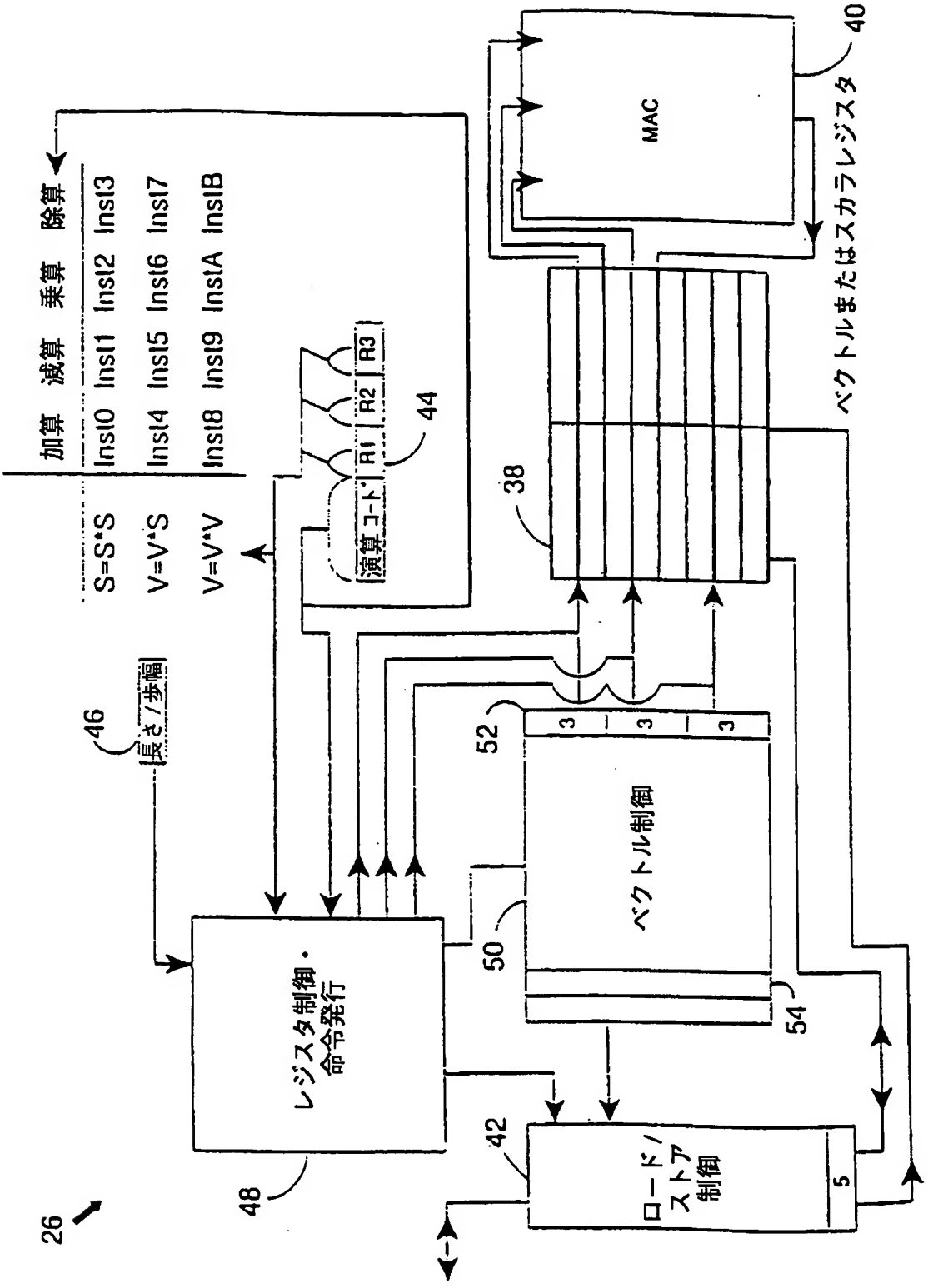
【図１７】

マルチタイタンプロセッサ内のレジスタバンクを示す図である。

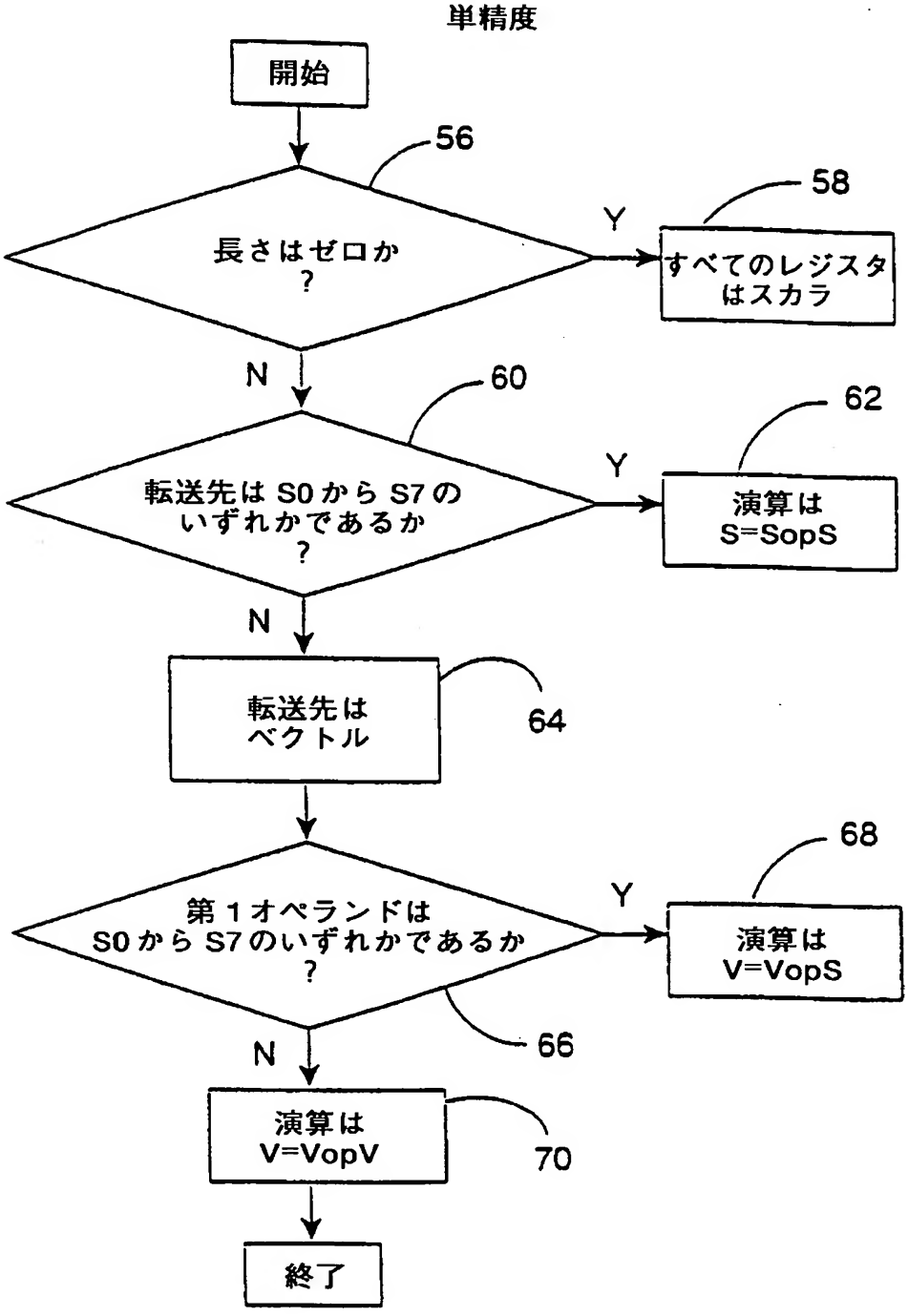
【図１】



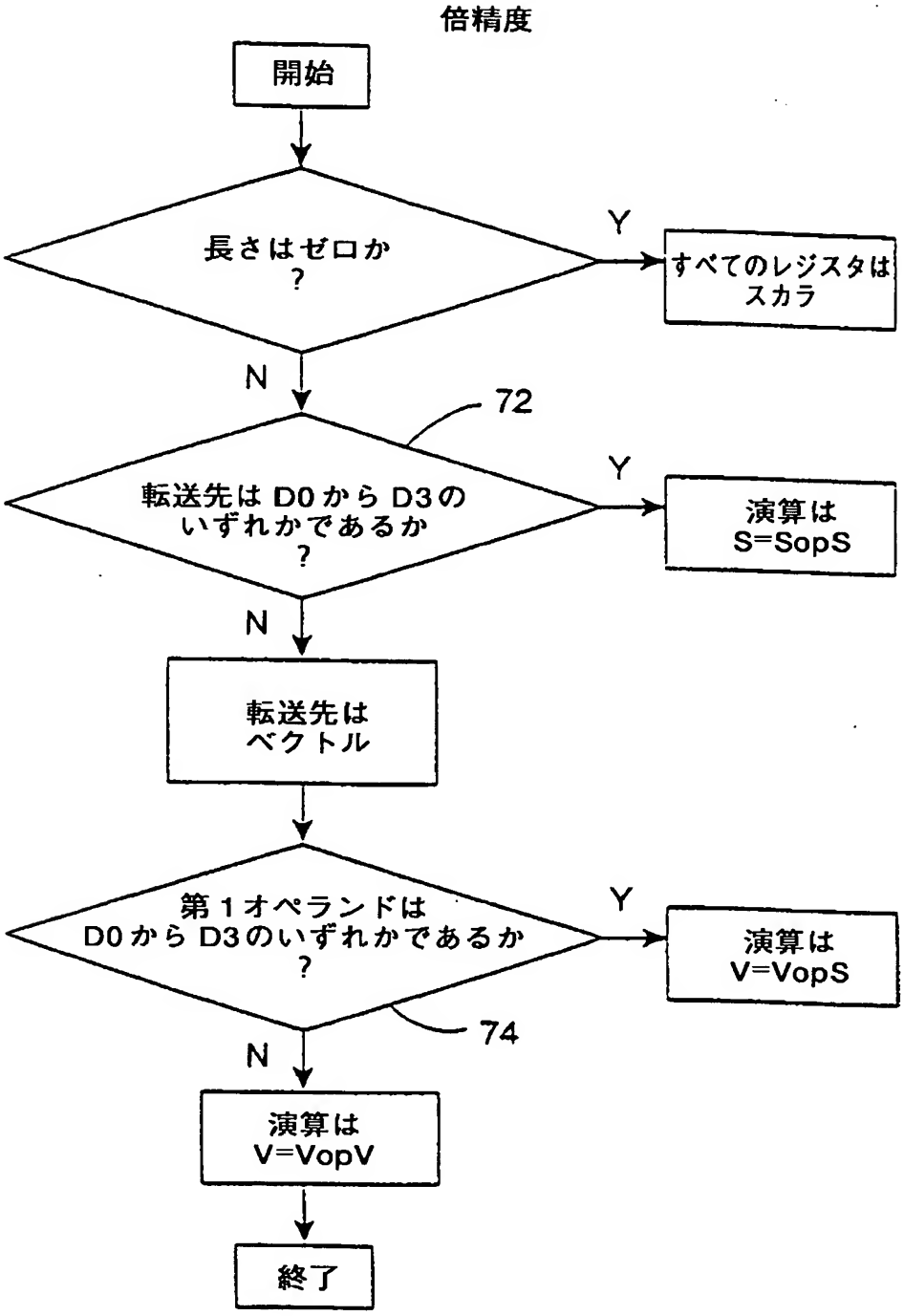
【図２】


加算 減算 乗算 除算
Inst0 Inst1 Inst2 Inst3
Inst4 Inst5 Inst6 Inst7
Inst8 Inst9 InstA InstB
S=S*S
V=V*S
V=V*V
演算コード
R1
R2
R3
44
46
長さ/歩幅
38
MAC
40
ベクトルまたはスカラレジスタ
52
3
3
3
ベクトル制御
50
54
レジスタ制御・命令発行
48
42
ロード/ストア制御
5
26

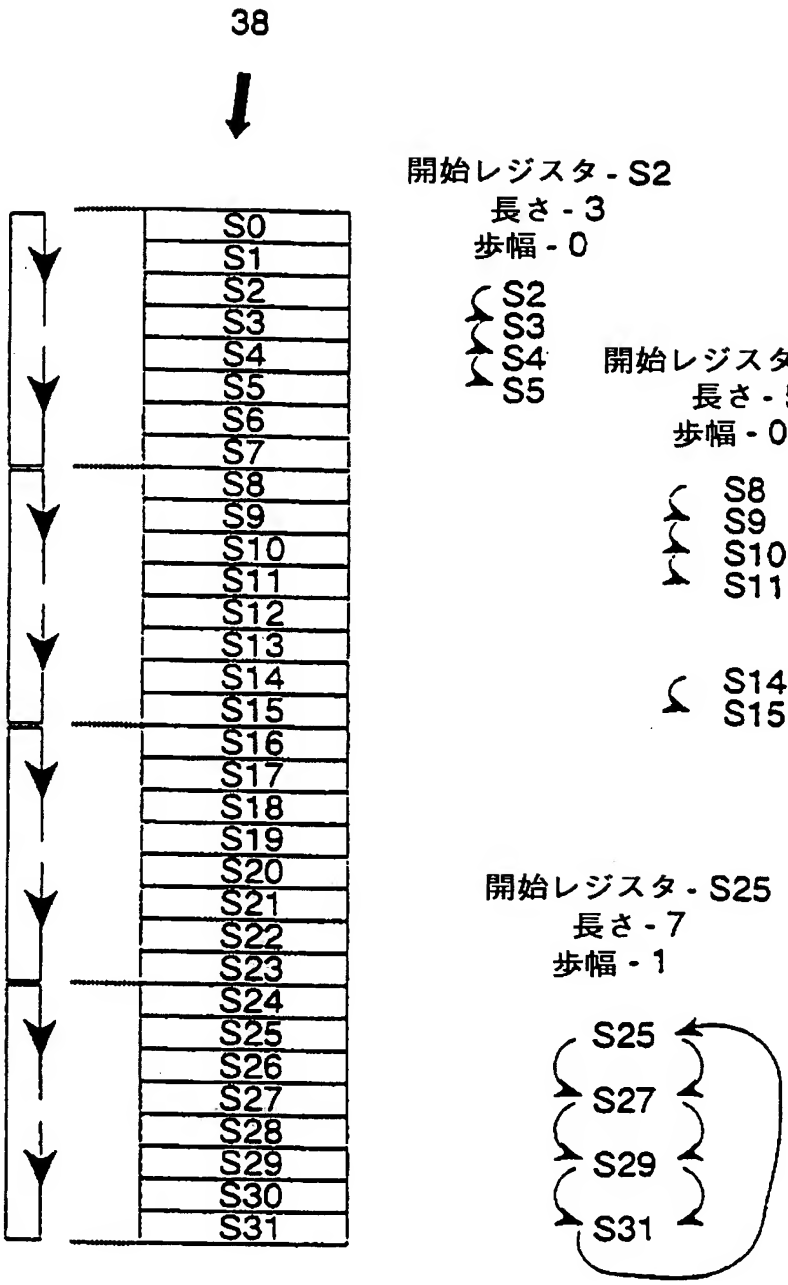
【図３】


単精度
開始
56
長さはゼロか
?
Y
58
すべてのレジスタ
はスカラ
N
60
転送先は S0 から S7 の
いずれかであるか
?
Y
62
演算は
S=SopS
N
転送先は
ベクトル
64
第１オペランドは
S0 から S7 のいずれかであるか
?
Y
68
演算は
V=VopS
66
N
演算は
V=VopV
70
終了

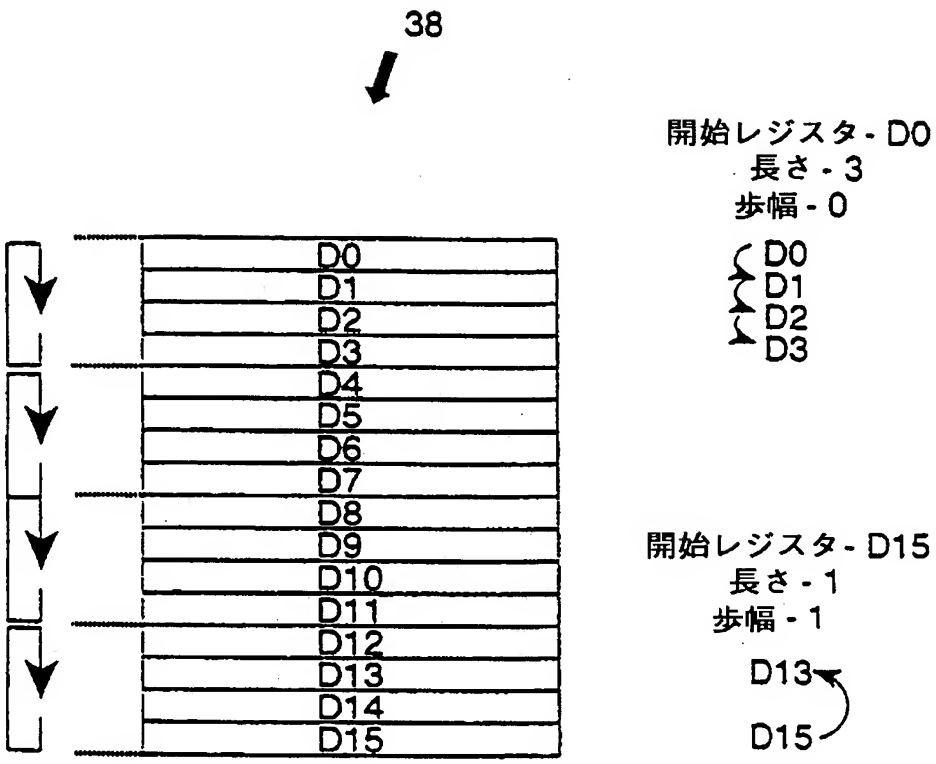
【図４】


倍精度
開始
長さはゼロか
?
Y
すべてのレジスタは
スカラ
N
72
転送先は D0 から D3 の
いずれかであるか
?
Y
演算は
S=SopS
N
転送先は
ベクトル
第１オペランドは
D0 から D3 のいずれかであるか
?
Y
演算は
V=VopS
74
N
演算は
V=VopV
終了

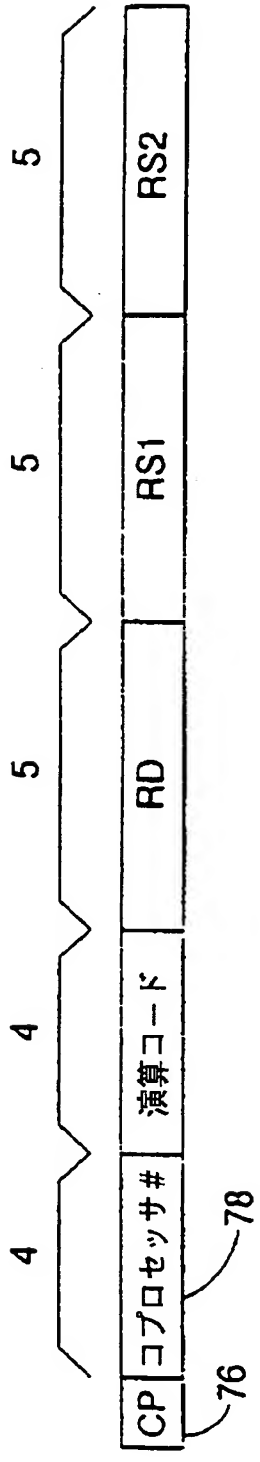
【図５】


38
S0
S1
S2
S3
S4
S5
S6
S7
S8
S9
S10
S11
S12
S13
S14
S15
S16
S17
S18
S19
S20
S21
S22
S23
S24
S25
S26
S27
S28
S29
S30
S31
開始レジスタ - S2
長さ - 3
歩幅 - 0
S2
S3
S4
S5
開始レジスタ - S14
長さ - 5
歩幅 - 0
S8
S9
S10
S11
S14
S15
開始レジスタ - S25
長さ - 7
歩幅 - 1
S25
S27
S29
S31

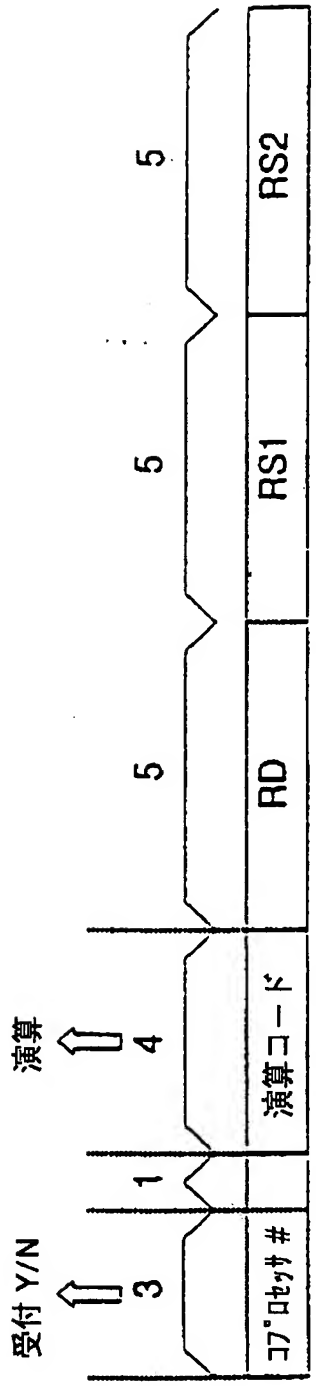
【図６】


38
開始レジスタ - D0
長さ - 3
歩幅 - 0
D0
D1
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11
D12
D13
D14
D15
開始レジスタ - D15
長さ - 1
歩幅 - 1
D13
D15

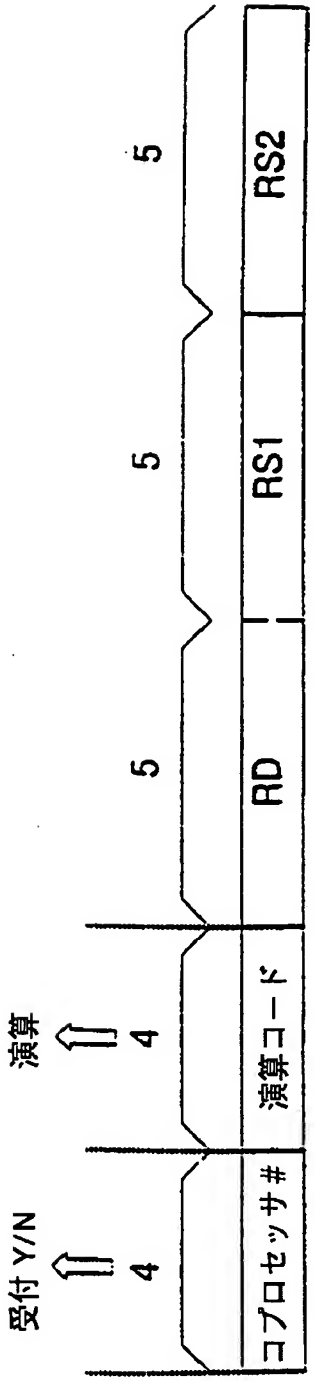
【図７Ａ】



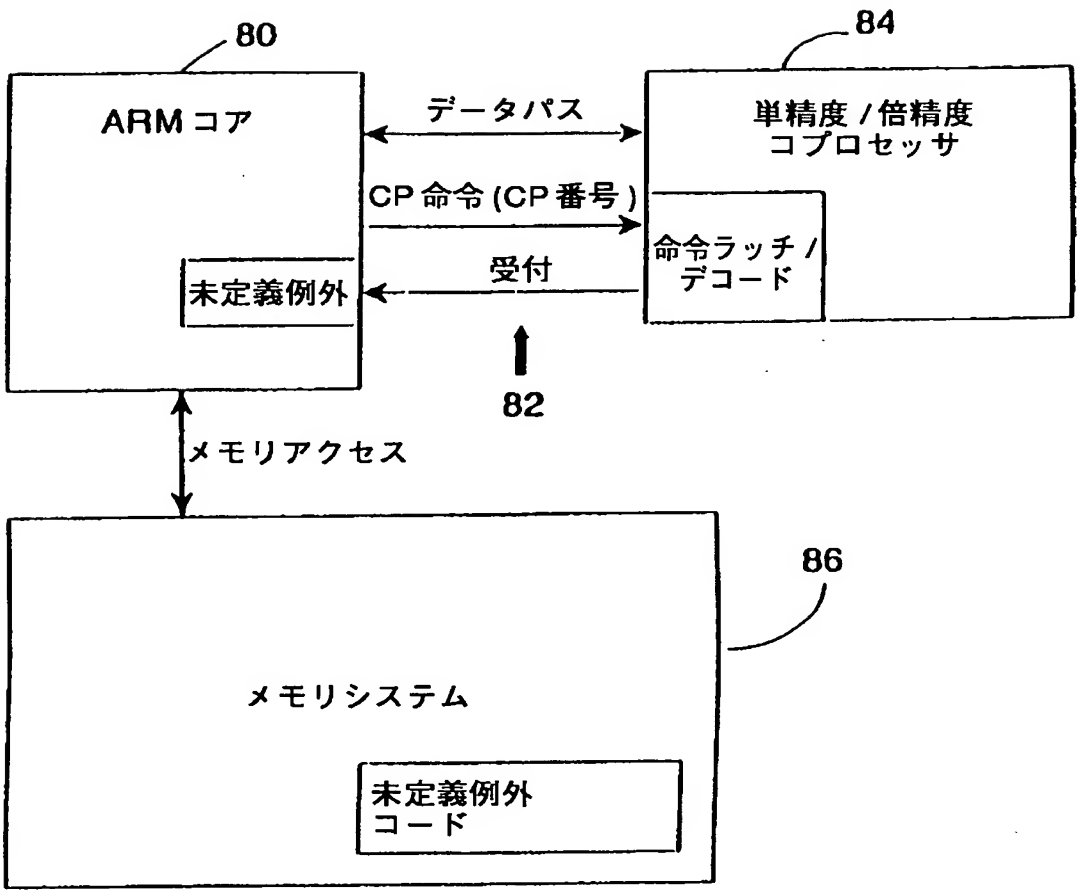
【図７Ｂ】



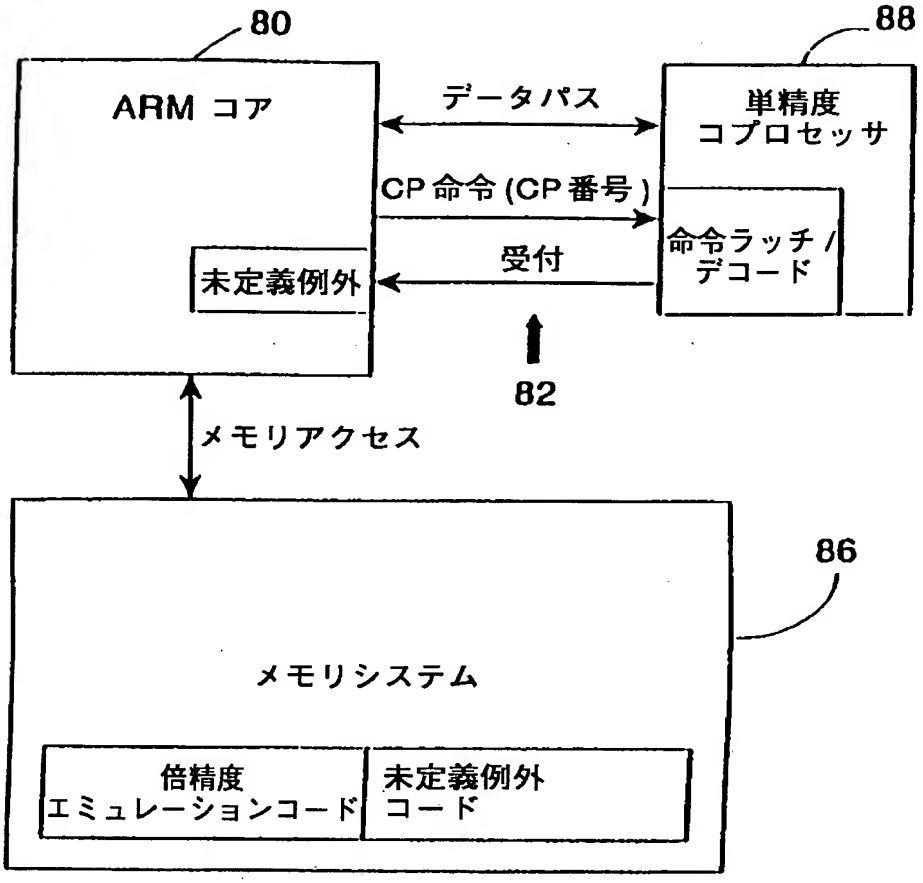
【図７Ｃ】



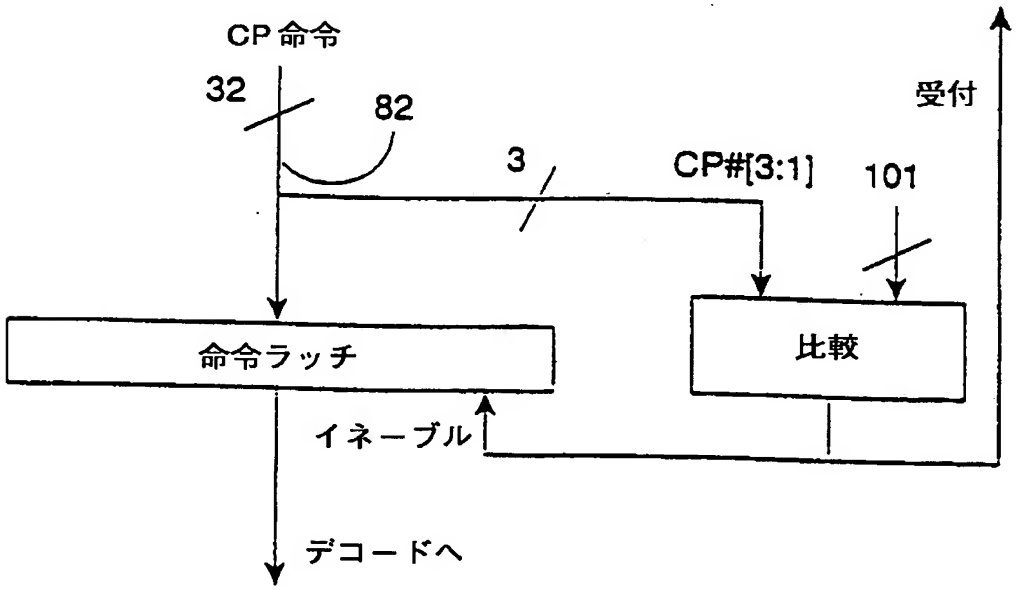
【図８】


80
ARMコア
データパス
CP命令(CP番号)
受付
未定義例外
84
単精度/倍精度
コプロセッサ
命令ラッチ/
デコード
82
メモリアクセス
86
メモリシステム
未定義例外
コード

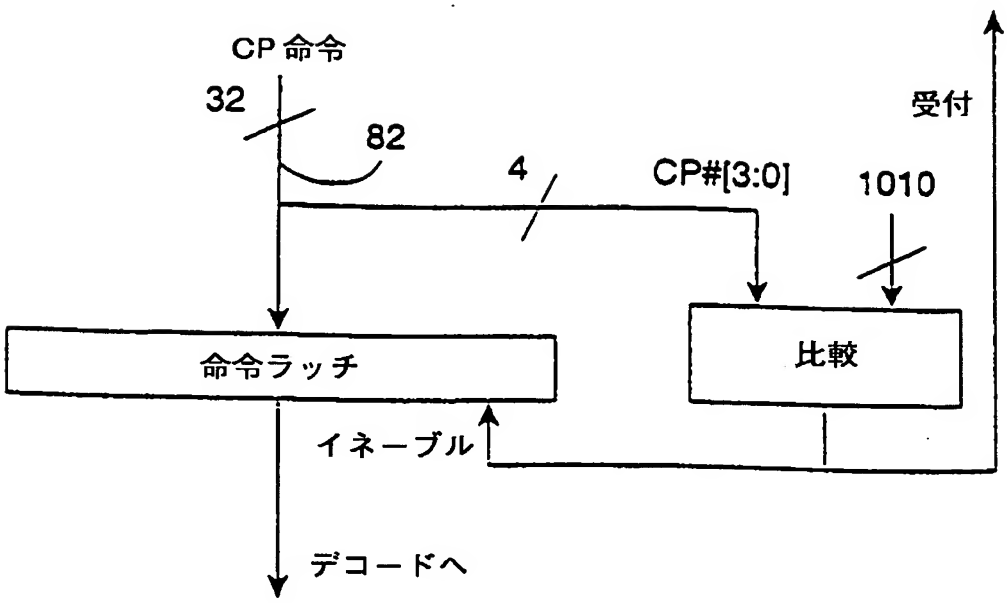
【図９】


80
88
ARM コア
データパス
単精度
コプロセッサ
CP 命令 (CP 番号)
命令ラッチ /
デコード
受付
未定義例外
82
メモリアクセス
86
メモリシステム
倍精度
エミュレーションコード
未定義例外
コード

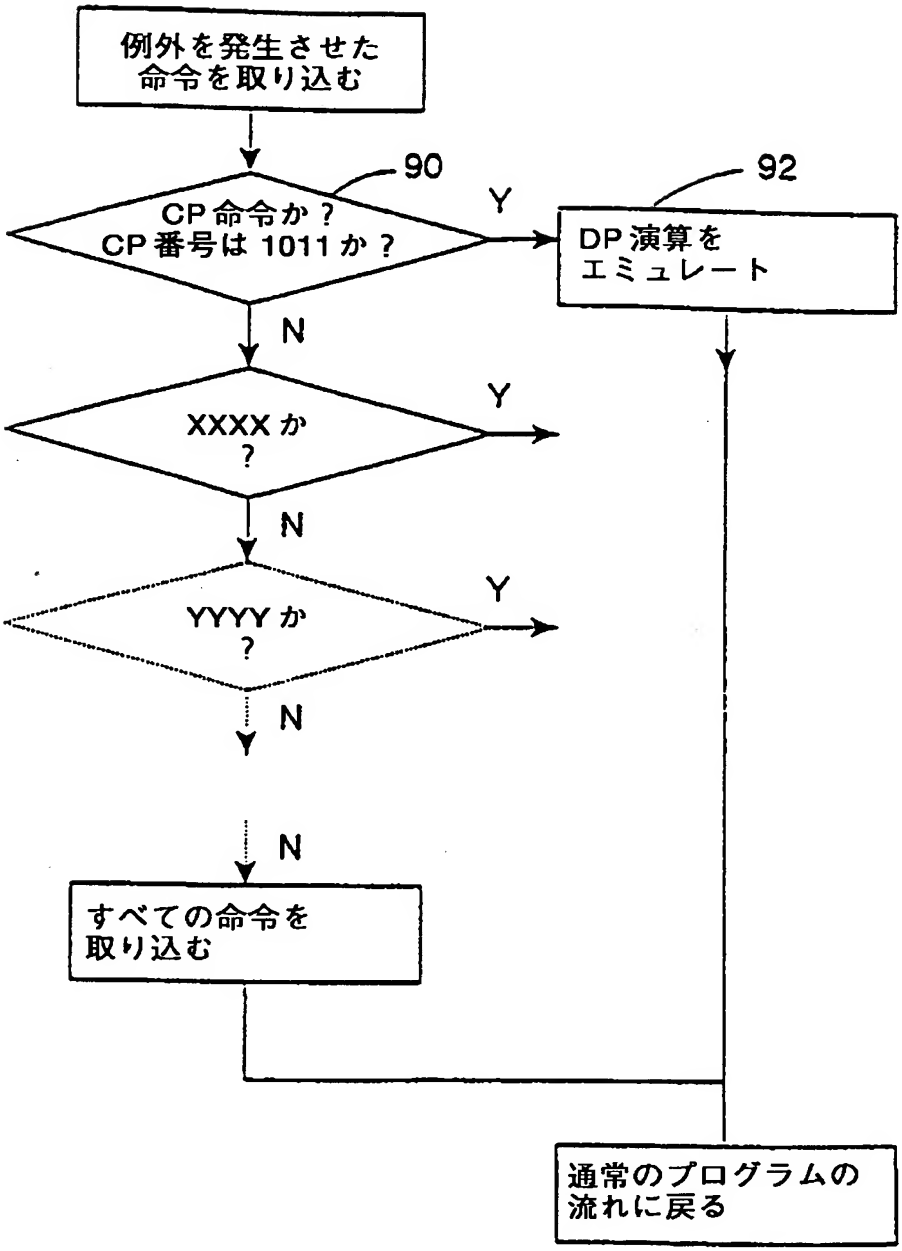
【図１０】


CP 命令
32
82
3
CP#[3:1]
101
受付
命令ラッチ
比較
イネーブル
デコードへ

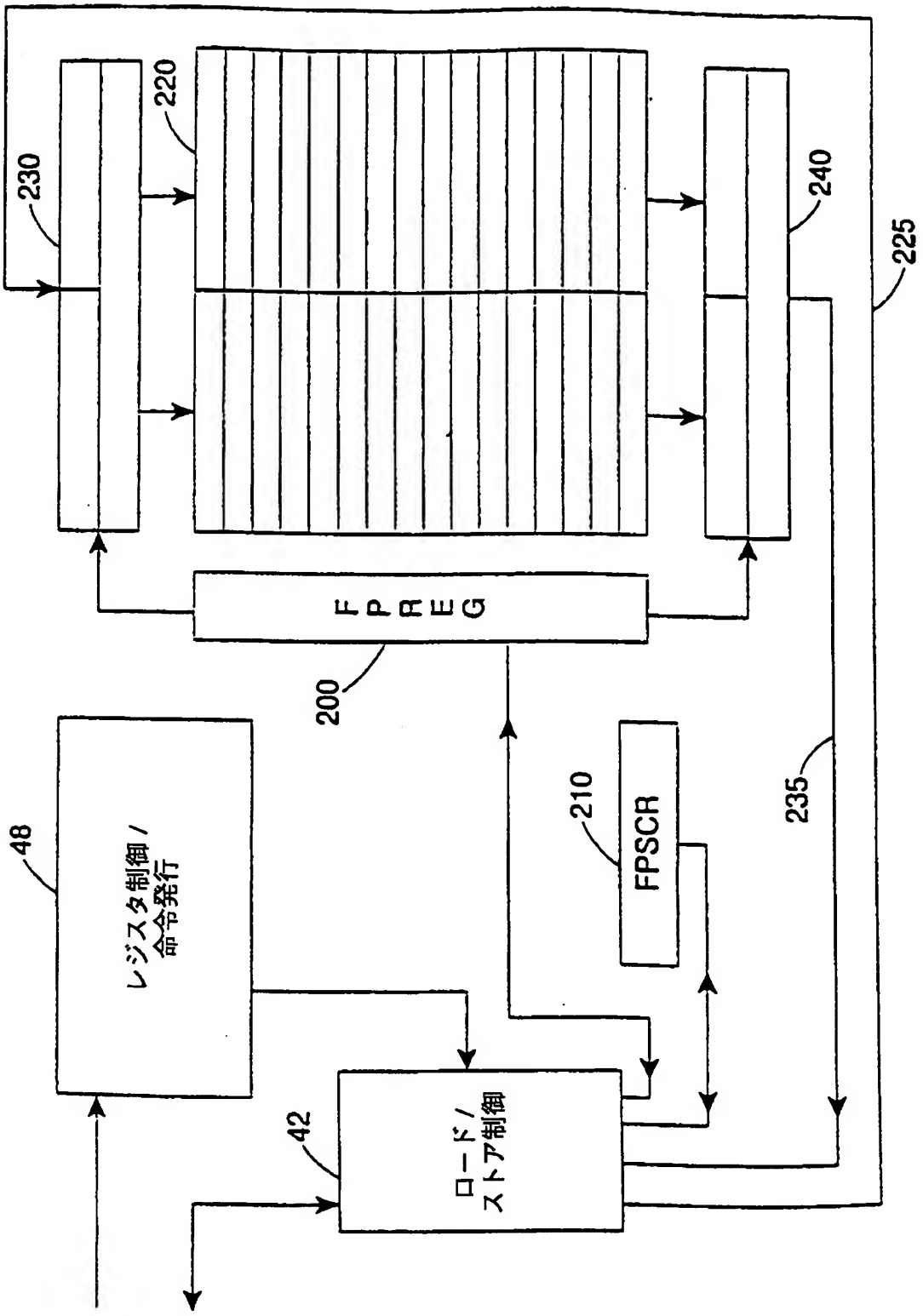
【図１１】



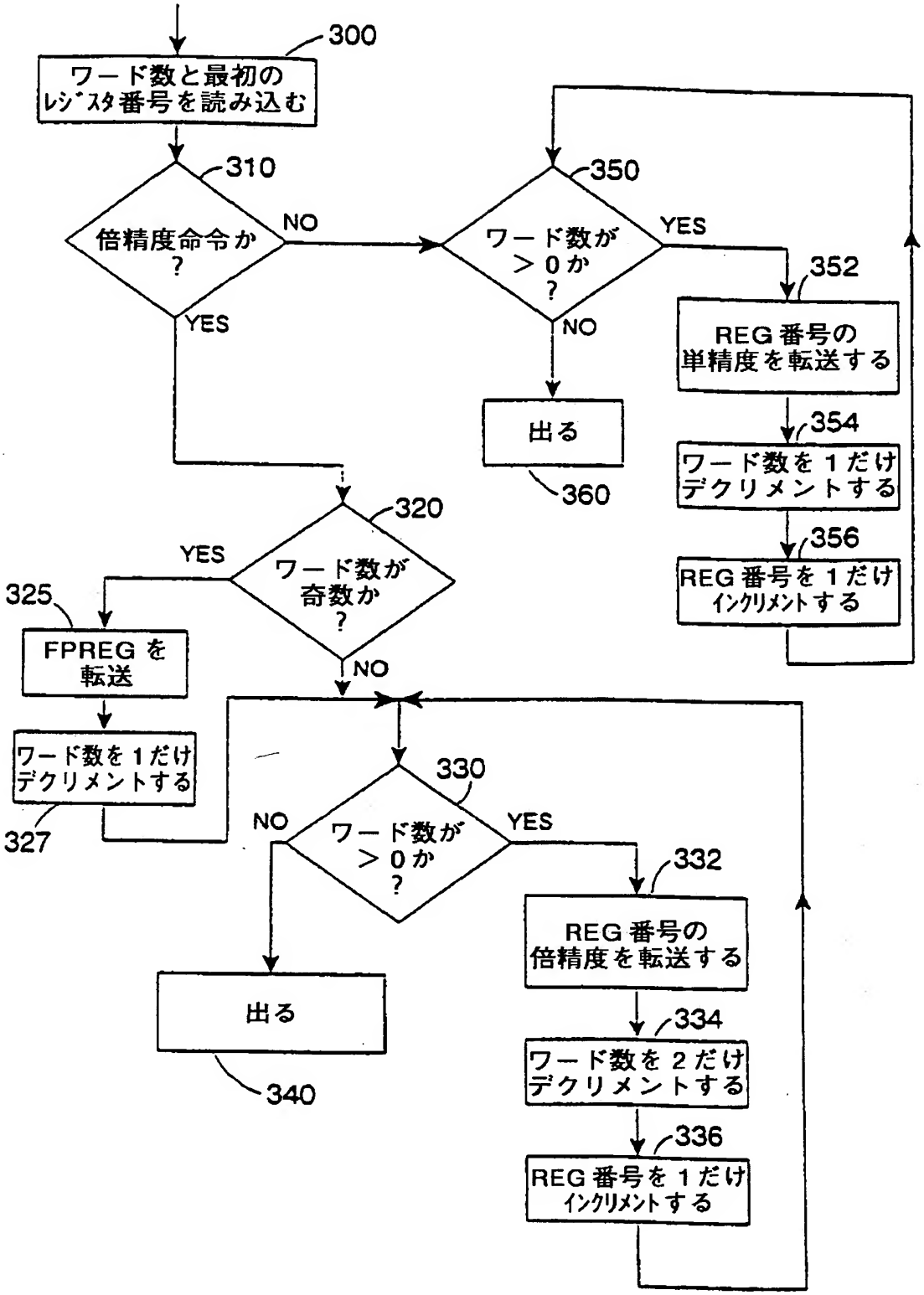
【図１２】


例外を発生させた
命令を取り込む
90
CP命令か？
CP番号は1011か？
Y
92
DP演算を
エミュレート
N
XXXXか
？
Y
N
YYYYか
？
Y
N
N
すべての命令を
取り込む
通常のプログラムの
流れに戻る

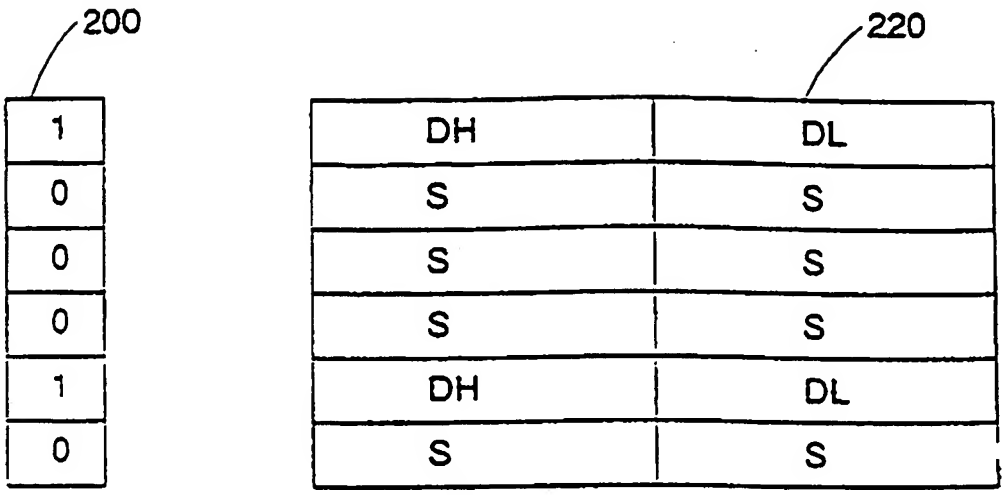
【図１３】


220
230
240
225
FPREG
200
210
FPSCR
235
48
レジスタ制御／
命令発行
42
ロード／
ストア制御

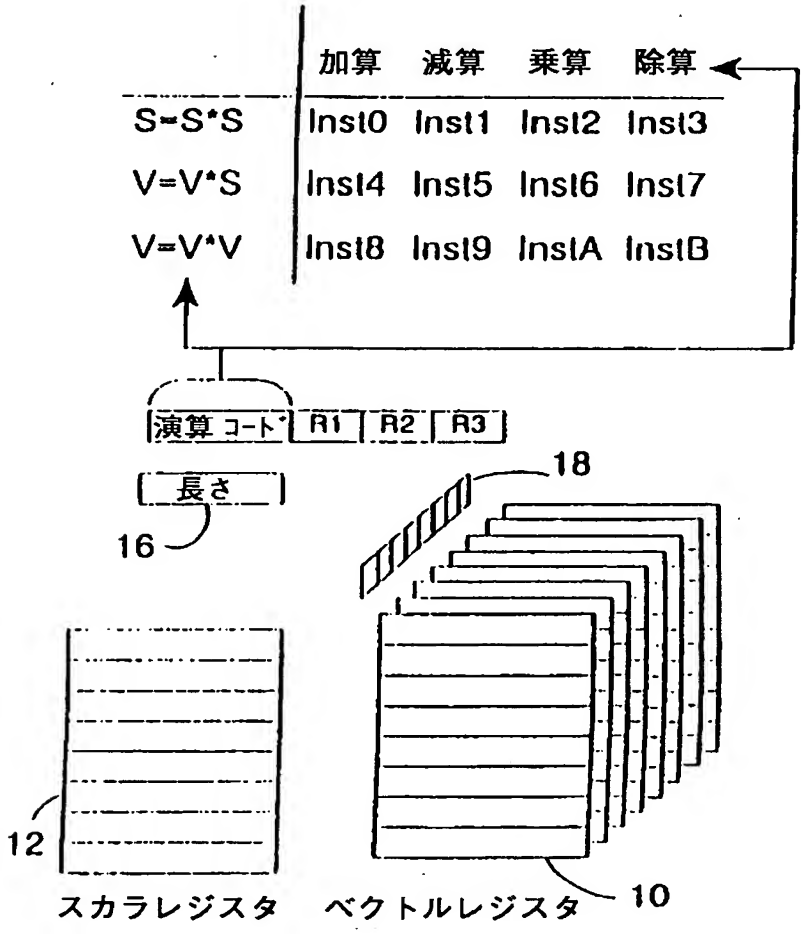
【図１４】


300
ワード数と最初の
レジスタ番号を読み込む
310
倍精度命令か
?
NO
YES
350
ワード数が
＞０か
?
YES
NO
352
REG番号の
単精度を転送する
354
ワード数を１だけ
デクリメントする
356
REG番号を１だけ
インクリメントする
出る
360
320
ワード数が
奇数か
?
YES
NO
325
FPREGを
転送
ワード数を１だけ
デクリメントする
327
330
ワード数が
＞０か
?
NO
YES
332
REG番号の
倍精度を転送する
334
ワード数を２だけ
デクリメントする
336
REG番号を１だけ
インクリメントする
出る
340

【図１５】



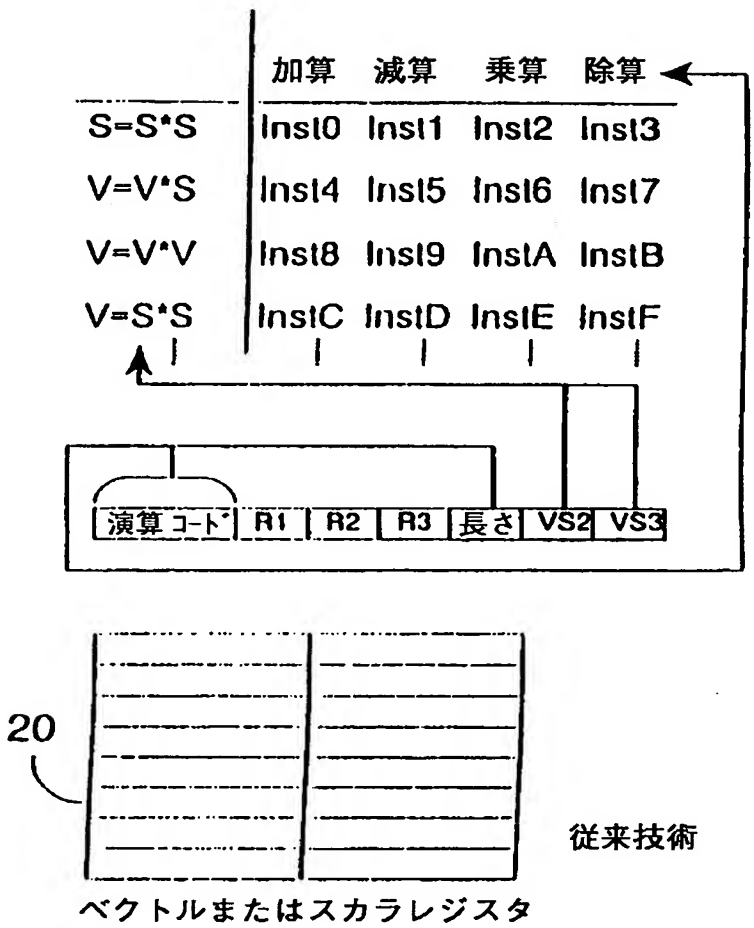
【図１６】



従来技術

クレイ１

【図１７】



DEC マルチタイタン

【手続補正書】特許協力条約第３４条補正の翻訳文提出書

【提出日】平成１２年３月９日（２０００．３．９）

【手続補正１】

【補正対象書類名】明細書

【補正対象項目名】請求項１

【補正方法】変更

【補正内容】

【請求項１】　データ処理装置であって、

アドレスでアクセス可能な複数のレジスタを有するレジスタバンクと、

前記レジスタバンク内の一連のレジスタのデータ値を使って前記データ処理命令に指示された初期レジスタから始めて、データ処理演算を複数回実行するベクトル演算を指示する少なくとも１つのデータ処理命令に応動する命令デコーダとを備え、

前記レジスタバンクは少なくとも１つのレジスタサブセットを含み、前記一連のレジスタが前記サブセット内に存在し、

前記命令デコーダは前記一連のレジスタが前記レジスタサブセット内で回り込むように前記一連のレジスタを制御する、ことを特徴とするデータ処理装置。

【手続補正２】

【補正対象書類名】明細書

【補正対象項目名】請求項１２

【補正方法】変更

【補正内容】

【請求項１２】　レジスタバンクのアドレスでアクセス可能な複数のレジスタ内にデータ値を格納するステップと、

ベクトル演算を指示する少なくとも１つのデータ処理命令に応動して、前記レジスタバンク内の一連のレジスタのデータ値を使って前記データ処理命令に指示された初期レジスタから始めて、データ処理演算を複数回実行するステップを有するデータ処理方法であって、

前記レジスタバンクは少なくとも１つのレジスタサブセットを含み、前記一連

のレジスタが前記サブセット内に存在し、

前記実行中に、前記一連のレジスタは前記レジスタサブセット内で回り込む、ことを特徴とするデータ処理方法。

【手続補正３】

【補正対象書類名】明細書

【補正対象項目名】０００９

【補正方法】変更

【補正内容】

【０００９】

１つの観点によれば、本発明によるデータ処理装置は、

アドレスでアクセス可能な複数のレジスタを有するレジスタバンクと、

前記レジスタバンク内の一連のレジスタのデータ値を使って前記データ処理命令に指示された初期レジスタから始めて、データ処理演算を複数回実行するベクトル演算を指示する少なくとも１つのデータ処理命令に応動する命令デコーダとを備え、

前記レジスタバンクは少なくとも１つのレジスタサブセットを含み、前記一連のレジスタが前記サブセット内に存在し、

前記命令デコーダは前記一連のレジスタが前記レジスタサブセット内で回り込むように前記一連のレジスタを制御する、ことを特徴とする。

【手続補正４】

【補正対象書類名】明細書

【補正対象項目名】００１０

【補正方法】変更

【補正内容】

【００１０】

レジスタバンクの（すべてのレジスタ数よりも少ない）レジスタ数のレジスタサブセット内で回り込むようにしたので、データ値を再ロードまたは移動することなくレジスタバンク内のデータ値を再使用するコンパクトなコードを書くことができる。一連のベクトルレジスタを分割するための余分な命令を用いることな

く必要な回り込みをハードウェアで行うことにより、命令コードは使用する度にサブセット内の異なった点から開始することができ、したがって異なった順序でデータ値を処理することができる。さらに、それら自身に対して回り込むレジスタのサブセットに対してベクトル演算を実行することにより、サブセット内に存在しない複数のレジスタのデータ値に対するデータ転送を同時に実行することができる。レジスタの回り込みはまた、例えば、バッファを互いにぐるぐる追いかけあう複数の点において、データをバッファに取り込みバッファから読み出すようにしたリング（循環）バッファ型構成をサポートするハードウェアを提供することによっても得られる。

【手続補正５】

【補正対象書類名】明細書

【補正対象項目名】００２０

【補正方法】変更

【補正内容】

【００２０】

別の観点によれば、本発明によるデータ処理方法は、

レジスタバンクのアドレスでアクセス可能な複数のレジスタ内にデータ値を格納するステップと、

ベクトル演算を指示する少なくとも１つのデータ処理命令に応動して、前記レジスタバンク内の一連のレジスタのデータ値を使って前記データ処理命令に指示された初期レジスタから始めて、データ処理演算を複数回実行するステップを有するデータ処理方法であって、

前記レジスタバンクは少なくとも１つのレジスタサブセットを含み、前記一連のレジスタが前記サブセット内に存在し、

前記実行中に、前記一連のレジスタは前記レジスタサブセット内で回り込む、ことを特徴とする。

【国際調査報告】

INTERNATIONAL SEARCH REPORT

International Application No
P[illegible]/GB 99/00707

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F15/78 G06F9/30

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	GB 2 216 307 A (ARDENT COMPUTER CORP) 4 October 1989 see page 7, line 7 - page 8, line 30 see page 13, line 7 - page 14, line 2	1-15
Y	EP 0 646 877 A (FUJITSU LTD) 5 April 1995 see page 10, line 17 - line 29	1-15
A	US 5 513 366 A (AGARWAL RAMESH C ET AL) 30 April 1996 see column 10, line 13 - line 61 see column 11, line 24 - column 12, line 13 see column 12, line 43 - line 60	1,12

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
6 July 1999	13/07/1999

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Daskalakis, T

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

International application No.
PCT/GB 99/08707

Box I Observations where certain claims were found unsearchable (Continuation of item 1 of first sheet)

This International Search Report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the International Application that do not comply with the prescribed requirements to such an extent that no meaningful International Search can be carried out, specifically:

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box II Observations where unity of invention is lacking (Continuation of item 2 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

see additional sheet

1. ☐ As all required additional search fees were timely paid by the applicant, this International Search Report covers all searchable claims.

2. ☒ As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this International Search Report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this International Search Report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest.

☐ No protest accompanied the payment of additional search fees.

International Application No. PCT/GB 99/00707

FURTHER INFORMATION CONTINUED FROM PCT/ISA/ 210

This International Searching Authority found multiple (groups of) inventions in this international application, as follows:

1. Claims: 1-7,9-15

Apparatus for processing data with register bank comprising a plurality of subsets of sequenses of registers.

2. Claim : 8

Apparatus for processing data with register bank comprising at least one subset of registers and a transfer controller for controlling transfers of data between the memory and said register bank

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/GB 99/00707

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
GB 2216307 A	04-10-1989	DE 3906327 A	14-09-1989
		FR 2628237 A	08-09-1989
		JP 2010467 A	16-01-1990
EP 0646877 A	05-04-1995	JP 7152733 A	16-06-1995
		US 5669013 A	16-09-1997
US 5513366 A	30-04-1996	NONE	

Form PCT/ISA/210 (patent family annex) (July 1992)

フロントページの続き

(72)発明者 マセニイ、デビッド、ターランス
アメリカ合衆国 テキサス、オースチン、バックソーン ドライブ 10706
(72)発明者 シール、デビッド、ジェームズ
イギリス国 ケンブリッジ、ケインズ ロード 68

Ｆターム(参考) 5B033 BA00 CA18 DD04
5B056 AA05 BB31 CC01 FF03 FF05
FF07 FF10 FF15 HH03 HH05

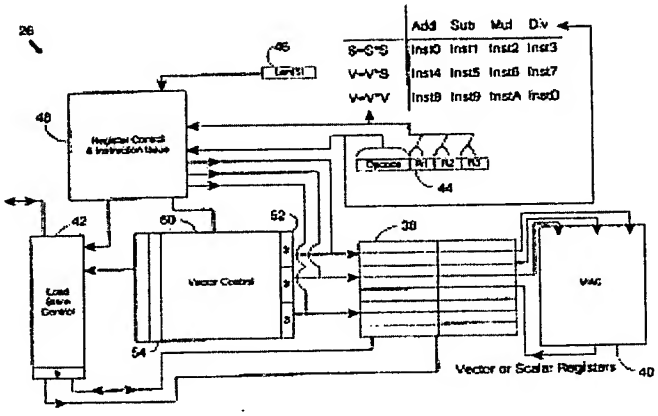
Recirculating register file

Publication number: JP2002517038T
Publication date: 2002-06-11
Inventor:
Applicant:
Classification:
- international: ***G06F9/34; G06F9/38; G06F17/16;*** **G06F9/34; G06F9/38; G06F17/16;** (IPC1-7): G06F17/16; G06F9/34
- european: G06F9/38E; G06F9/38H; G06F9/38P
Application number: JP20000551329T 19990309
Priority number(s): WO1999GB00707 19990309; US19980085752 19980527

Also published as:
EP1080421 (A1)
US6189094 (B1)
EP1080421 (A0)
EP1080421 (B1)
RU2225995 (C2)
more >>

Abstract not available for JP2002517038T
Abstract of corresponding document: **US6189094**

A floating point unit having a register bank containing a plurality of registers supports vector operations that execute a specified operation a plurality of times upon a sequence of data values form different registers. The register bank is divided into subsets and with the sequence of registers used in a vector operation wrapping within a subset. The subsets comprise disjoint, contiguous ranges of register numbers. The wrapping within ranges allows compact code and efficient to be provided for performing DSP operations, such as FIR filtering and matrix transformations.



* NOTICES *

1.This document has been translated by computer. So the translation may not reflect the original precisely.
2.**** shows the word which can not be translated.
3.In the drawings, any words are not translated.

CLAIMS

[Claim(s)]
[Claim 1] It is a data processor. The register bank which has two or more registers, It has an instruction decoder following at least one data-processing instruction which directs the vector operation which carries out multiple-times activation of the data-processing operation using the data value of a series of registers in said register bank. Said register bank contains at least one register subset. Said a series of registers exist in said subset. Said instruction decoder is a data processor characterized by what said a series of registers are controlled for so that said a series of registers turn within said register subset.
[Claim 2] Said vector operation performs said data-processing operation using two or more data values to which the register of two or more reams corresponds. The register of said two or more reams exists in each subset including the register subset of plurality [register bank / said]. Said instruction decoder is equipment according to claim 1 characterized by what the register of said two or more reams is controlled for so that the register of said two or more reams turns within each register subset.
[Claim 3] Said two or more subsets are relatively prime equipment according to claim 2 characterized by things.
[Claim 4] Said subset is equipment given in either claim 1 characterized by what it has for one register group who consists of registers with which a number was assigned continuously, claim 2 and claim 3.
[Claim 5] Each of two or more of said subsets is equipment according to claim 2 characterized by what it has for one register group who consists of registers with which a number was assigned continuously.
[Claim 6] Said two or more subsets are equipment according to claim 5 characterized by what it has for two or more register groups whom each register group who consists of a register with which a number was assigned continuously is following.
[Claim 7] Said two or more subsets are equipment according to claim 6 characterized by what it has four continuing register groups for.
[Claim 8] It is equipment according to claim 1 to 7 which has further the transfer controller which controls a transfer of the data value between memory, and said memory and register in said register bank, and is characterized by what said transfer controller transmits a series of data values for following two or more move instructions between said memory and a series of registers in said register bank.
[Claim 9] Each register group is equipment according to claim 6 characterized by what is accessed in the address through the incrementer which performs a surroundings lump among the register group's both ends.
[Claim 10] Equipment according to claim 1 to 9 characterized by what said a series of registers are a series of continuous registers.
[Claim 11] Said register bank and said instruction decoder are equipment according to claim 1 to 10 characterized by what is been a part of floating-point unit.
[Claim 12] The step which stores a data value in the register of the plurality of a register bank, At least one data-processing instruction which directs vector operation is followed. It is the data-processing approach of having the step which carries out multiple-times activation of the data-processing operation using the data value of a series of registers in said register bank. As for said register bank, said a series of registers exist in said subset including at least one register subset. It is the data-processing approach characterized by what said a series of registers turn around within said register subset during said activation.
[Claim 13] Said vector operation performs said data-processing operation using two or more data values to which the register of two or more reams corresponds. The register of said two or more reams exists in each subset including the register subset of plurality [register bank / said]. It is the data-processing approach according to claim 12 characterized by what the register of said two or more reams turns around within said register subset during said activation.

[Claim 14] It is the data-processing approach according to claim 13 which the data value in a series of registers is the tap multiplier of a filter, and is characterized by what the data value in a series of another registers is a signal value by which a filter is carried out with said filter.

[Claim 15] The data-processing approach according to claim 12 characterized by what two or more vector operations are performed for to the data value in said two or more Muraji's register by changing the start point of a series of registers for every vector operation at least.

[Translation done.]

* NOTICES *

1.This document has been translated by computer. So the translation may not reflect the original precisely.
2.**** shows the word which can not be translated.
3.In the drawings, any words are not translated.

DETAILED DESCRIPTION

[Detailed Description of the Invention]
[0001]
This invention relates to the field of data processing. This invention relates to the data processing system which has a register bank and supports vector operation in more detail.
[0002]
Offering the data processing system which has a register bank and supports vector operation is known. As an example of such a system, there are clay 1 and a multi-tie tongue processor of DEC.
[0003]
Clay 1 processor has a vector register bank and the Scala register bank separately. When the operation code of the instruction currently executed shows vector operation, a series of data values are returned from a vector register bank according to the mask stored in the die-length value and mask register which are stored in the die-length register. Specifying how many data values die length has in the data value of a series of, a mask specifies which data value is returned out of two or more data values corresponding to the vector register shown in the instruction.
[0004]
A multi-tie tongue processor has one register bank, and the register in it is used as Scala or a vector. the register specified as the instruction itself -- Scala -- or the flag which shows a vector, and the die-length field which shows the number of the data values in said a series of data values when a vector register is used are included.
[0005]
The vector instruction itself is desirable. It is because two or more data-processing operations by single instruction can be specified, so code density can be raised. Digital signal processing, such as an audio and graphics operation, is suitable for especially using vector operation. Activation of the filter operation which applies the tap multiplier of a digital filter to a series of signal values etc. is because the demand which performs the same operation to the data value to which a single string relates occurs frequently.
[0006]
Moreover, it is desirable to perform a data-processing operation efficiently as quickly as possible. One policy which gathers a rate and effectiveness is avoiding again loading or the need of positioning again for the data value already stored in the register bank. In case this is realized, the instruction code which can carry out the reuse of the data value has the problem that it tends to become for a long time and complicated. If many instructions are needed directing a required operation, processing will become slow and the purpose of carrying out the reuse of the data value in a register bank will serve as an invalid.
[0007]
Instead of using general-purpose processors, such as clay 1 and a multi-tie tongue processor, giving the special function which supports a small number of digital-signal-processing operation to the digital digital disposal circuit of the special purpose is often performed. A data value required in big memory is stored in the digital digital disposal circuit of these special purposes, and a general approach takes out a data value required for each operation if needed. It is not necessary to reload a data value in big memory or, and it does not have to carry out repositioning. It is because it is controlled by operating the address used in order that those use sequence might access big memory. This approach has the problem acquired when a circuit must be specially designed so that it may have consistency in the operation performed, therefore a more typical general-purpose processor is used that the flexibility and ease of integration with other functions are missing.
[0008]

The purpose of this invention is offering efficient and quick data processing, using a register bank and the instruction decoder which is supporting vector operation, and maintaining the flexibility of a general-purpose processor.

[0009]

According to one viewpoint, the data processor by this invention Register bank which has two or more registers It has an instruction decoder following at least one data-processing instruction which directs the vector operation which carries out multiple-times activation of the data-processing operation using the data value of a series of registers in said register bank. Said register bank contains at least one register subset. Said a series of registers exist in said subset. Said instruction decoder is characterized by what said a series of registers are controlled for so that said a series of registers turn within said register subset.

[0010]

Since it was made to turn within the register subset of the number of registers (fewer than all the numbers of registers) of a register bank, the compact code which carries out the reuse of the data value in a register bank can be written without reloading or moving a data value. By performing a required surroundings lump by hardware, a data value can be processed in sequence which could start instruction code from the point of having differed in the subset whenever it used it, therefore is different, without using the excessive instruction for dividing a series of vector registers. Furthermore, data transfer to the data value of two or more registers which do not exist in a subset can be performed to coincidence by performing vector operation to the subset of a register around which it turns to these selves. A surroundings lump of a register is obtained also by offering the hardware which supports the ring (circulation) buffer mold configuration which incorporates data to a buffer and was made to carry out multiplication from the buffer again in the point which pursues the buffer of each other round and round, and suits.

[0011]

Although the subset of a surroundings lump register can also be set only to one, to consider as the following systems is more advantageous. That is, said vector operation performs said data-processing operation using two or more data values to which two or more Muraji's register corresponds. Said two or more Muraji's register exists in each subset including the register subset of plurality [register bank / said]. Said instruction decoder controls said two or more Muraji's register so that said two or more Muraji's register turns within each register subset.

In a digital-signal-processing operation, it is desirable for the need of carrying out the reuse of the data value from the register of 2 ream to use two or more surroundings lump register subsets by that (for example, the FIR operation or matrix operation which those offset is changed and does the multiplication and accumulation of a tap and a signal value) which is often generated.

[0012]

Although a subset may lap, since it is usually separated from the required data value of the reuse in such a situation in fact, a subset is good also as relatively prime. By this, the implementation of hardware becomes simpler and it is convenient.

[0013]

Two or more subsets can also consist of registers in the location mixed with the register which does not exist in these subsets. However, if two or more subsets are the groups of a ******** register with a number continuously, programming and an implementation will become easy.

[0014]

To continue is more desirable although the register group of each other is isolable in a register bank. It is because the register tooth space which can use this gentleman can be used more efficiently.

[0015]

The capacity of this invention which uses vector operation more effectively is complemented in the following desirable example. That is, it is equipment characterized by what it has further the transfer controller which controls a transfer of the data value between memory, and said memory and register in said register bank, and said transfer controller transmits a series of data values for following two or more move instructions between said memory and a series of registers in said register bank.

[0016]

The capacity of this invention to use vector operation efficiently according to the capacity to deliver a data value to the register block in a register bank since the block can be exchanged with one instruction after carrying out the reuse of the register block several times is acquired.

[0017]

Dividing using a series of registers in vector operation and a register bank into the register subset which was

able to be defined beforehand can be efficiently carried out in the desirable example to which each register group is characterized by what is accessed in the address through the incrementer which performs a surroundings lump among the register group's both ends.
[0018]
A series of registers used for vector operation can take many gestalten, such as a register in every other one in a subset. However, most generally a useful gestalt is a gestalt which are a series of registers with which a series of registers continued.
[0019]
The above-mentioned approach can be used for any processors which have a register bank and support vector operation. However, it also turns out that it does not interfere in other consideration adopted in the example to which, especially as for the capacity which uses a code as a compact and carries out the reuse of the data value in a register, it turns out to which that it is useful, and the register bank and the instruction decoder exist in a floating-point unit.
[0020]
According to another viewpoint, the data-processing approach by this invention The step which stores a data value in the register of the plurality of register bank, At least one data-processing instruction which directs vector operation is followed. It is the data-processing approach of having the step which carries out multiple-times activation of the data-processing operation using the data value of a series of registers in said register bank. As for said register bank, said a series of registers exist in said subset including at least one register subset. It is characterized by what said a series of registers turn around within said register subset during said activation.
[0021]
This approach is useful especially when performing efficiently the FIR filter operation which the relative offset between a tap multiplier value and a signal value is changed for every vector operation, and carries out the reuse of these values several times.
The example of this invention is explained with reference to the following drawings as one example.
[0022]
Drawing 1 shows data processing system 22, and this contains a main processor 24, the floating-point unit co-processor 26, cache memory 28, main memory 30, and input/output system 32. A main processor 24, cache memory 28, main memory 30, and input/output system 32 are connected through Maine Bath 34. The co-processor bus 36 connects a main processor 24 to the floating-point unit co-processor 26.
[0023]
On the occasion of actuation, a main processor 24 (it is also called an ARM core) performs the data-processing instruction train which controls a general data-processing operation including a dialogue with cache memory 28, main memory 30, and input/output system 32. The co-processor instruction is embedded in the data-processing instruction train. A main processor 24 recognizes these co-processor instructions as what should be performed by the attached co-processor. Therefore, a main processor 24 is received from the co-processor bus 36 by the co-processor of attachment of a co-processor instruction by sending these co-processor instructions on the co-processor bus 36. The floating-point unit co-processor 26 receives and executes the co-processor instruction which detected being turned to itself. This detection is performed through the co-processor number field inside a co-processor instruction.
[0024]
Drawing 2 shows the floating-point unit co-processor 26 more to a detail typically. the floating-point unit co-processor 26 -- 32 32-bit registers (shown in drawing 2 few) from -- the becoming register bank 38 is included. These registers can operate as a register of the pair which operated according to the individual as a single precision register with which each stored 32 bit-data value, or stored 64 bit-data value by two. In the floating-point unit co-processor 26, the ****** unit 40 and the load store control unit 42 of pipeline control are prepared. In a suitable situation, the ****** unit 40 and the load store control unit 42 of pipeline control operate to coincidence, the ****** unit 40 of pipeline control performs arithmetic operation (power accumulation and other operations are included) to the data value in a register bank 38, and the load store control unit 42 delivers the data value which the ****** unit 40 of pipeline control is not using to the floating-point unit co-processor 26 through a main processor 24.
[0025]
In the floating-point unit co-processor 26, the received co-processor instruction is latched in an instruction register 44. In this simplified drawing, it is possible that a co-processor instruction consists of the three register directions fields R1, R2, and R3 (these fields may be divided and the whole instruction may be

made to distribute them variously in practice) following operation code and it. These register directions fields R1, R2, and R3 support the register in the register bank 38 which functions as the destination, the 1st source, and the 2nd source of the data-processing operation currently performed, respectively. The vector control register 46 (some big registers are sufficient as this rather than it performs an additional function) stores the die-length value and step value for the vector operation performed in floating-point unit co-processor 26 grade. The vector control register 46 is initialized according to a vector control register load instruction, and can be updated with a die-length value and a step value. Since a vector die-length value and a step value are globally used within the floating-point unit co-processor 26, these values can be dynamically changed with the global base, without depending on a self-modification code.
[0026]
It is possible that register control and the instruction issue unit 48, the load store control unit 42, and the vector control unit 50 perform a part for the principal part of the function of an instruction decoder jointly. Register control and the instruction issue unit 48 output an initial register access (address) signal to a register bank 38 without decoding to operation code, or without using the vector control unit 50 according to operation code and the three register directions fields R1, R2, and R3. Thus, since direct access can be carried out to an initial register value, early activation is attained. If a vector register is directed, the vector control unit 50 will make a series of register access signals using the triplet incrementer (adder) 52. The vector control unit 50 accesses a register bank 38 in the address corresponding to the die-length value and step value which were stored in the vector control register 46. The register scoreboard 54 is formed in order to perform a register lock. This is for the ****** unit 40 of pipeline control and the load store control unit 42 which operates to coincidence not to cause a data consistency problem (the register scoreboards 54 can also be considered to be a part of register control and instruction issue unit 48).
[0027]
The operation code in an instruction register 44 specifies the classification (which [, such as addition, subtraction, multiplication, a division, loading, and a store,] is instructions?) of the data-processing operation performed. This is not related to whether the register specified is a vector or Scala. Decoding of an instruction and the setup of the ****** unit 40 are made easy by this. The 1st register indicated value R1 and the 2nd register indicated value R2 are cooperation, and code the vector / Scala classification of the operation specified by operation code. Three common cases supported by coding are S=S*S (for example, fundamental random count created by the C compiler from the block of the C code), V=V op S (for example, carry out enlarging or contracting of the element of a vector), and V=V op V (for example, matrix operations, such as an FIR filter and graphic conversion) (in the above-mentioned sentence, "op" is a general operation and functor is a destination = second-operand op first operand). Moreover, depending on an instruction (for example, instruction in comparison with zero or an absolute value), it does not have a destination register (an output is a condition flag), or there is a thing with an insufficient (the instruction in comparison with zero has only one input operand) input operand. In such a case, since options, such as a vector / Scala classification, are specified, many operation code bit tooth spaces are usable, and can make all registers usable at each operand (for example, a compare instruction may completely [whatever the register] always be Scala).
[0028]
Although register control and the instruction issue unit 48, and the vector control unit 50 perform a part for the principal part of the function of an instruction decoder together, they determine and control the vector / Scala classification of the directed data-processing operation according to the 1st register indicated value R1 and the 2nd register indicated value R2. When the die-length value stored in the vector control register 46 shows 1 (it corresponds to the value of the stored zero), this can be used as early directions of pure scalar operation.
[0029]
Drawing 3 is a flow chart and shows the flow of the processing used in order to decode a vector / Scala classification from register indicated value in single precision mode. In step 56, it investigates whether vector die length is globally set as 1 (it is equivalent to die-length value zero). When vector die length is 1, in step 58, all registers are treated as Scala. In step 60, the destination register R1 investigates whether it is within the limits of S0 to S7. When that is right, all operations are Scala, and as shown in step 62, they serve as constructor form voice of S=S op S. When step 60 is NO, as shown in step 64, it is determined that the destination is a vector. When the destination is a vector, a code is treated noting that a second operand is also with a vector. Therefore, two possibility which remains in this phase is V=V op S and V=V op V. The check of step 66 which investigates whether a first operand is either of S0 to S7 performs distinction of

these two options. If that is right, operations are V=V op S, otherwise, V=V op V. These conditions are recognized at steps 68 and 70, respectively.
[0030]
When vector die length is set as 1, all of 32 registers of a register bank 38 can be used as Scala. It is because the Scala classification of an operation is identified at step 58, without depending on the check of step 60 which will restrict the number of the registers which can be used for the destination. When the vector and the Scala instruction use them, combining, the check of step 60 is useful although all Scala instructions are identified. Moreover, when calculating in the mixture mode of a vector and Scala, supposing a first operand is Scala, it is either of S0 to S7. Supposing a first operand is a vector, it will be either of S8 to S31. It is adaptation to a thing large generally to offer 3 times of the usable number of registers in a register bank to the first operand which is a vector, when the number of registers required in order to hold a series of data values uses vector operation.
[0031]
It will be understood that the general operation which a user wants to perform is graphic conversion. The conversion performed can be expressed with 4*4 matrix in a common case. That the reuse of the operand is carried out to such count shows that it is desirable to store a matrix value to the register treated as a vector. Similarly, an input pixel value is usually stored in four registers which can be treated as a vector so that a reuse can be carried out. The output of matrix operation is Scala (what accumulated separate vector line multiplication) usually stored in four registers. The vector register of 24 (16+4+4) individuals and the Scala register of eight (4+4) individual are needed to treat a twice as many input value as this and an output value.
[0032]
Although drawing 4 is a flow chart corresponding to drawing 3 , double precision mode is shown in this case. As stated above, in double precision mode, the register slot in a register bank 38 functions as a pair, and stores 64 bit-data value of 16 pieces in D15 from the logic register D0. In this case, the code of the vector / Scala classification of a register is drawing 3 , and is changed. That is, it has changed to whether "whether the destination to be either of D0 to D3" and the check "whether a first operand is either of D0 to D3", respectively. [in / in the check of steps 60 and 66 / steps 72 and 74]
[0033]
Coding the vector / Scala classification of the register in the register appointed field as mentioned above produces the difficulty of case some of subtraction or a non-changing operation like a division, although order bit space is saved sharply. When a register configuration is V=V op S, about the problem which lacks symmetry between the first operand of a non-changing operation, and a second operand, it can conquer by extending an instruction set by taking in the pair of operation codes, such as SUB and RSUB showing two different operand options of non-commutative operation and DIV, and RDIV, without exchanging a register value with an additional instruction.
[0034]
Drawing 5 shows how to rotate a vector in the subset of a register bank 38. Especially in single precision mode, a register bank is divided into four register range, and those addresses are S0 to S7 and S8 to S15 and S16 to S23 and S24 to S31. These register range is continuing without a common element mutually. In drawing 2 , the surroundings lump function of these subsets that have eight registers can be offered by using the triplet incrementer (adder) 52 into the vector control unit 50. If the subset range is crossed, an incrementer will turn back. This simple actuation becomes easy by adjusting the subset of two or more range which consists of 8 words in a register address tooth space.
[0035]
Drawing 5 is referred to again. In order to help an understanding of return actuation of a register, some vector operations are shown. As for the first vector operation, 4 (shown by 3 which is a die-length value in the vector control register 46), and a step value specify [the initiation register / S2 and vector die length] 1 (shown by the zero which are a step value in the vector control register 46). Therefore, when an instruction is decoded where these global vector control parameter is set up, and referring to a register S2 as a vector, this instruction is executed 4 times, using respectively the data value in registers S2 and S3, S4, and S5. Since this vector does not cross the subset range, a surroundings lump of a vector is not performed.
[0036]
For an initiation register, in the 2nd example, S14 and vector die length are [6 and a step value] 1. In this case, an instruction will begin from a register S14 and will be executed 6 times. Next, the register used is S15. A register's increment of only a step value rotates the register used to the register S8 instead of S16 shortly. An instruction is executed further 3 times and completes all the processes S14, S15, and S8, S9, and

S10 and S11.
[0037]
For an initiation register, in the example of the last of drawing 5 , S25 and vector die length are [8 and a step value] 2. The register used first is S25 and the degree is S27, S29, and S31 according to a step value. After using a register S31, the following register value turns back, passes a register S24 at the beginning of a subset according to return and the step value 2, and performs an operation using a register S25. In case an incrementer 52 moves between vector registers, it is good at the triplet adder which applies a step value to the current value. Therefore, a step can be adjusted by supplying a step value which is different in an adder.
[0038]
Drawing 6 shows a surroundings lump of a register bank 38 in double precision mode. In this mode, the subset of a register consists of D0 to D3 and D4 to D7 and D8 to D11 and D12 to D15. The minimum value inputted into the adder which functions as an incrementer 52 in double precision mode is 2. This is equivalent to 1 which is the step of double precision. When the step of double precision is 2, it is necessary to input 4 into an adder. For an initiation register, in the first example shown in drawing 6 , D0 and vector die length are [4 and a step value] 1. In this case, the sequence of a vector register is D0, D1, D2, and D3. Since the range of a subset is not crossed, there is no surroundings lump in this example. For an initiation register, in the 2nd example, D15 and vector die length are [2 and a step value] 2. In this case, the sequence of a vector register is D15 and D13.
[0039]
In drawing 2 , the load store control unit 42 has a 5-bit incrementer in the output, and the circumference lump of a register by which loading/store multiplex operation is applied to vector operation is not performed. Single loading/store multiplex instruction can access the register with which a required number continues by this.
[0040]
There is an FIR filter divided into the unit which consists of four signal values and four taps as an example of the operation using a surroundings [this] lump configuration. When syntax R8-R11 op R16-R19 express the vector operation of R8opR16, R9opR17, R10opR18, and R11opR19, an FIR filter operation can be performed as follows.
Eight taps to R8-R15 load 8 signal value to R16-R23 R8-R11opR16-R19 -- and put a result into R24-R27 R9-R12opR16-R19 -- and accumulate a result in R24-R27 R10-R13opR16-R19 -- and accumulate a result in R24-R27 R11-R14opR16-R19 -- and accumulate a result in R24-R27 Load a new tap to R8-R11 again. R12-R15opR16-R19 -- and accumulate a result in R24-R27 R13-R8opR16-R19 -- and accumulate a result in R24-R27 (R15-> it turns to R8) R14-R9opR16-R19 -- and Accumulate a result in R24-R27 (R15-> it turns to R8). Accumulate R15-R10opR16-R19 and a result in R24-R27 (R15-> it turns to R8). Load a new tap to R12-R15 again. When a tap is lost load new data to R16-R19 again R12-R15opR20-R23 -- and put a result into R28-R31 R13-R8opR20-R23 -- and accumulate a result in R28-R31 (R15-> it turns to R8) R14-R9opR20-R23 -- and accumulate a result in R28-R31 (R15-> it turns to R8) Accumulate R15-R10opR20-R23 and a result in R28-R31 (R15-> it turns to R8). the remainder -- the above -- the same .
[0041]
As mentioned above, loading can be performed [therefore] to juxtaposition to a register which is different from two or more accumulating totals (that is, double buffering can be attained).
[0042]
As for drawing 7 A, a main processor 24 shows typically how a co-processor instruction is seen. A main processor identifies an instruction as a co-processor instruction using the field 76 (this may divide) which is the combination of the bit in an instruction. A co-processor instruction includes the co-processor number field 78 in a standard ARM processor instruction set. A co-processor connected to the main processor investigates whether it is that by which the specific co-processor instruction was addressed to these co-processors using this co-processor number field 78. A co-processor number which is different in co-processors of a different type, such as a DSP co-processor (for example, Il Piccolo co-processor made by ARM) or a floating-point unit co-processor, can be assigned, therefore it can access in the address separately within one system using the same co-processor bus 36. A co-processor instruction contains the operation code which a co-processor uses again, and three bit fields [five]. These 5 bit field specifies the destination, a first operand, and a second operand out of a co-processor register, respectively. A main processor enables it to perform a data-processing operation with both desirable co-processors and main processors by decoding a co-processor instruction partially at least in some instructions, such as a co-processor load and a store. A main processor can also decode the data type coded in the co-processor number as part of instruction

decoding performed in such a situation again.
[0043]
Drawing 7 B shows the case where the co-processor which supports double precision and single-precision operation decodes the received co-processor instruction. Two continuous co-processor numbers are assigned to such a co-processor, and it checks whether itself is the co-processor of the destination using the top triplet of a co-processor number. Thus, since the least significant bit of a co-processor number turns into an excessive bit when checking the co-processor of the destination, it can use in order to specify the data type used in this in case the co-processor instruction is executed. In this example, a data type corresponds to the data size of single precision or double precision.
[0044]
In double precision mode, it turns out that the number of registers decreases from 32 to 16 substantially. Therefore, although register field size can be made small, decoding of which register to use in that case cannot be directly obtained from the self-inclusion field in the known location in a co-processor instruction, but is dependent on decoding of other parts of a co-processor instruction. This not only complicates actuation of a co-processor, but has disadvantageous profit of making it late. By coding a data type using the least significant bit of a co-processor number, operation code stops being completely dependent on a data type, and will simplify the decoding, and it will speed up.
[0045]
Drawing 7 C shows the case where the co-processor which supports only the single data type which is the subset of the data type which the co-processor of drawing 7 B supports decodes a co-processor instruction. In this case, it is decided whether that instruction should be received using a perfect co-processor number. Thus, if it is the data type by which the co-processor instruction is not supported, since it corresponds to another co-processor number, it will not be received. And a main processor 24 performs instruction exception handling of the undefined, and emulates an operation to the data type which is not supported.
[0046]
Drawing 8 shows the data processing system which has the ARM core 80. The ARM core 80 functioned as a main processor, and is connected with the co-processor 84 which supports both single precision and a double precision data type through the co-processor bus 82. If the co-processor instruction containing a co-processor number is found out within an instruction train, it will be passed on the co-processor bus 82 from the ARM core 80. And if a co-processor 84 is in agreement in a co-processor number as compared with the number of itself, it will send a reception signal to the ARM core 80. Supposing it does not receive a reception signal, it will recognize that an ARM core is an undefined-instruction exception, and the exception-handling code stored in the memory system 86 will be referred to.
[0047]
Drawing 9 shows the system which exchanged the co-processor 84 in the system of drawing 8 to the co-processor 88 which supports only single precision operation. In this case, a co-processor 88 recognizes only one co-processor number. Therefore, although the double precision co-processor instruction in the original instruction train is executed depending on the co-processor 84 of drawing 8 , it is not received depending on the single precision co-processor 88. Therefore, the undefined exception-handling code in a memory system 86 can include a double precision emulation routine to perform the same code.
[0048]
Although the need of emulating a double precision instruction makes activation of these instructions late, the single precision co-processor 88 is advantageous, when it can do cheaply small rather than the double precision co-processor 84 and a double precision instruction does not appear rarely enough.
[0049]
Drawing 10 shows the instruction latch circuit in the co-processor 84 which supports both single precision and a double precision instruction, and has two adjoining co-processor numbers. In this case, top triplet CP [of the co-processor number in a co-processor instruction] # [3:1] is compared with the number assigned to the co-processor 84. In this example, when the co-processor 84 has the co-processor numbers 10 and 11, this comparison can be performed by testing top triplet CP[of a co-processor number] # [3:1] by comparison to a binary number 101. Supposing it is in agreement, a reception signal will be returned to the ARM core 80, and a co-processor instruction will be latched and executed.
[0050]
Drawing 11 shows the equal circuit in the single precision co-processor 88 of drawing 9 . In this case, only one co-processor number is recognized and single precision operation is used as a default. The comparison which performs whether a co-processor instruction should be received and latched when deciding is

performed between all 4-bit CP# [3:0] of a co-processor number, and the binary digit 1010 which is one embedded co-processor number.
[0051]
Drawing 12 shows the flow in the case of starting the undefined exception-handling routine of the example of drawing 9 , and running a double precision emulation code. Therefore, the following procedures are completed. It investigates whether the instruction made to generate an undefined-instruction exception is a co-processor instruction which has the binary number 1011 which is a co-processor number (step 90). If that is right, this instruction is meant as a double precision instruction, and can be emulated at step 92. Then, it returns to the flow of a main program. When other exception types are not detected at step 90, they can also detect and process at a previous step.
[0052]
Drawing 13 shows the example which used the format register FPREG200 for storing the information which identifies the type of the data stored in the 32-bit each register or data slot of a register bank 220. Each data slot can make it operate separately as a single precision register for storing 32 bit-data value (data word), as stated above. Or it can also be made to operate as a double precision register for making it other data slots and a pair and storing 64 bit-data value (2 data word). According to the desirable example of this invention, the format register FPREG200 is constituted so that it may identify whether the data slot stores single precision data or double precision data in it.
[0053]
As shown in drawing 13 , 32 data slots of a register bank 220 are constituted so that 16 pairs of data slots may be offered. When the first data slot stores the single precision data value in it, since the data slot of the another side of a pair of stores only a single precision data value and a double precision data value is stored in a desirable example, it is not linked with other data slots. this -- which data slot pair -- two single precision data values -- or either of one double precision data value will certainly be stored. This information is discriminable using the 1-bit information in connection with each data slot pair in a register bank 220. Therefore, in a desirable example, the format register FPREG200 is constituted so that the 16-bit information that the type of the data stored in each data slot pair of a register bank 220 is identified may be stored. Therefore, the format register FPREG200 can be constituted as a 16-bit register or a 32-bit register which has 16-bit information in order to maintain adjustment with other registers in the floating-point unit co-processor 26.
[0054]
Drawing 15 shows six pairs of data slots in a register bank 220. Since the double precision data value of six pieces or the single precision data value of 12 pieces is stored according to the desirable example, these data slot pairs can be used. The example of data storable in these data slot is shown in drawing 15 . DH expresses the 32 most significant bits of a double precision data value, DL expresses the 32 least significant bits of a double precision data value, and S expresses a single precision data value.
[0055]
The entry to which it corresponds in the format register FPREG200 by the desirable example of this invention is also shown in drawing 15 . According to the desirable example, the single precision data value is stored in at least one data slot of a data slot pair to which a value "0" corresponds by the value "1" stored in the format register FPREG200 showing that the double precision data value is stored in a corresponding data slot pair, or it means that both data slots are not initialized. That is, when both data slots are not initialized, one data slot is not initialized among data slot pairs, the data slot of another side stores the single precision data value or the data slot of the both of a pair of stores the single precision data value, a logic "0" value is stored in the bit to which the format register FPREG200 corresponds.
[0056]
the co-processor instruction which could use the FPU co-processor 26 of a desirable example, and could process either the single precision data value or the double precision data value, and the main processor 24 published as stated above -- it -- a single precision instruction -- or a double precision instruction (see drawing 7 B and the related description) is identified. When an instruction is received by the co-processor, register control and the instruction issue unit 48 are passed, and it decodes and performs. If an instruction is a load instruction, register control and the instruction issue unit 48 will direct to take out the data specified as the load store control unit 42 from memory, and to store it in the data slot as which the register bank 220 was specified. The data from which the co-processor is taken out in this phase will know a single precision data value or a double precision data value, and the load store control unit 42 operates according to it. Therefore, the load store control unit 42 passes either a 32-bit single precision data value or a 64-bit double

precision data value to the register bank input logic 230 through a path 225, and is made to store it in a register bank 220.
[0057]
Data enable it to identify whether each data slot pair which was supplied also to the format register FPREG200, added the required bit to it, and it is not only loaded to a register bank 220, but received data with the load store control unit 42 stores single precision data or double precision data. In a desirable example, before this data is loaded to a register bank, it is stored in the format register FPREG200. This is because the register bank input logic 230 can use this information.
[0058]
In a desirable example, the internal format of the data in a register bank 220 is the same as an external format. Therefore, a single precision data value is stored as a 32 bit-data value, and a double precision data value is stored in a register bank 220 as a 64 bit-data value. Since the register bank input logic 230 can access the format register FPREG200, the data which it has received now know single precision or double precision. Therefore, in such an example, the register bank input logic 230 only arranges data so that the data received through the path 225 may be stored in the suitable data slot of a register bank 220. However, in another example, when the internal format in a register bank differs from an external format, the register bank input logic 230 can also be constituted so that required conversion may be performed. For example, a numeric value is expressed as what applied the value to which only the characteristic usually carried out the exponentiation of the base value to 1.abc--. Since 1 on the left-hand side of decimal point is expressed, single precision typical for effectiveness and a double precision expression do not use a data bit. Rather, the 1 is contained tacitly. When the internal representation currently used in the register bank 220 needs to express 1 with a certain reason clearly, the register bank input logic 230 performs required data conversion. In such an example, in order to hold the additional data made by the register bank input logic 230, a data slot is usually partly larger than 32 bits.
[0059]
The load store control unit 42 not only loads a data value to a register bank 220, but can load data to one or more system registers FPSCR210 of a co-processor 26, for example, the user status and a control register. In a desirable example, the user status and a control register FPSCR210 contain the configuration bit and exception status bit which a user can access. This detail is left to the explanation of the architecture of a floating-point unit shown in the end of explanation of an example.
[0060]
When the data slot in the register bank 220 which has the contents from which register control and the instruction issue unit 48 should store store instruction in reception, and the instruction should store it in memory is specified, the actuation according to it is directed to the load store control unit 42, and required data word is read from a register bank 220 to the load store control unit 42 through the register bank output logic 240. The register bank output logic 240 accesses the contents of the FPREG register 200, in order that the data currently read may judge single precision data or double precision data. And by performing suitable data conversion, data conversion which the register bank input logic 230 performed is returned, and the data is supplied to the load store control logic 42 through a path 235.
[0061]
According to the desirable example of this invention, when store instruction is a double precision instruction, it is possible that the co-processor 26 is operating by the 2nd mode of operation which applies an instruction to a double precision data value. Since the double precision data value holds even data word, the store instruction published by the 2nd mode of operation specifies even data slots which usually have the contents which should be stored in memory. However, according to the desirable example of this invention, if odd data slots are specified, the load store control unit 42 reads the contents of the FPREG register 200, and after it stores these contents in memory first, it stores even data slots as which it was specified in the register bank 220. In order to specify the data slot which should usually be transmitted, the number of the data slots which begin from the data slot which specified the specific data slot in a register bank by the base address first, and was specified as the degree and which should be stored (namely, the number of data word) is specified numerically.
[0062]
Therefore, since the number of the specified data slots is odd although the contents of the data slot of a total of 32 pieces are stored in memory when store instruction gives the data slot of the beginning in a register bank 220 as a base address and specifies 33 data slots, for example, the contents of the FPREG register 200 are also stored in memory.

[0063]
By this approach, both the contents of the register bank and the contents of the FPREG register 200 which identifies the data type stored in the data slot of a register bank 220 are storable in memory using one instruction. Since the contents of the FPREG register 200 are clearly stored by this, the problem that another instruction must be published can be avoided, therefore it does not have a bad influence on memory into the instruction execution of loading from a store or memory at a process speed.
[0064]
In the further example of this invention, it is also storable in memory by advancing one more step of this technique using one instruction if needed, additional system register 210, for example, FPSCR register. The example of the register bank 220 which has 32 data slots examined above is considered. When 33 data slots are specified by store instruction, the FPREG register 200 is stored in memory with the contents of 32 data slots in a register bank 220. However, when different odd number exceeding the number of data slots in a register bank, for example, 35**, is specified, as for the load store control unit 42, this can be interpreted as the demand which should also store the contents of the FPREG register 200, and the not only the data slot of a register bank 220 but contents of the FPSCR register 210 in memory. A co-processor can also prepare the further system register, for example, the exception register which specifies the exception generated by the co-processor while processing the instruction. When different odd number in store instruction, for example, 37**, is specified, the load store control unit 42 can be interpreted as the demand in which the contents of one or more exception registers should also store this with the contents of the FPSCR register 210, the FPREG register 200, and the register bank 220.
[0065]
This technique has the case where the code which directs a store or a load instruction does not know the contents of the register bank, and the useful contents of a register bank, especially when [mere] it is temporarily stored in memory and is taken out by the register bank later. When the code knows the contents of the register bank, it may be necessary to store the contents of the FPREG register 200 in memory. The examples of representation of the code which does not know the contents of the register bank are a context change code, a procedure call entry, and exit routine.
[0066]
In such a case, as the contents of the FPREG register 200 can be efficiently stored in memory with the contents of the register bank and being stated in the top, to be sure, an alien-system register is also storable if needed.
[0067]
The same process will be performed if a consecutive load instruction is received. Therefore, if the double precision load instruction which specifies odd data slots is received, the load store control unit 42 loads the contents of the FPREG register 200 to the FPREG register 200, loads the contents of the system register shown with the number of slots specified by the load instruction, and stores even more data word in the data slot as which the register bank 220 was specified. Therefore, when it thinks in the example described above and the number of data slots specified by the load instruction is 33, the contents of a FPREG register are loaded to the FPREG register 200, and the contents of 32 data slots are loaded following it. When the number of data slots similarly specified by the load instruction is 35, not only the above-mentioned contents but the contents of the FPSCR register 210 are loaded to a FPSCR register. At the end, when the specified number of data slots is 37, not only the above-mentioned contents but the contents of the exception register are loaded to these exceptions register. Probably the specific actuation in connection with specific odd number will completely be arbitrary, and it will be clear for this contractor for it to be able to change if needed.
[0068]
Drawing 14 is the flow chart showing actuation of the register control and the instruction issue logic 48 by the desirable example of this invention when executing store instruction and a load instruction. First, the number of data word (this is the same as the number of data slots in a desirable example) is read from an instruction with the first register number specified with an instruction in step 300, i.e., a base register. And in step 310, it investigates whether this instruction is a double precision instruction. As stated above, this information is available to a co-processor in this phase. An instruction is because it specifies the double precision instruction or the single precision instruction.
[0069]
When an instruction is a double precision instruction, processing progresses to step 320 and the numbers of words specified with an instruction there judge whether it is odd number. In this example, it assumes that the

technique for transmitting a system register alternatively with the FPREG register 200 does not use it, when numbers of words are odd, this shows what the contents of the FPREG register 200 should be transmitted for, therefore in step 325, the contents of the FPREG register are transmitted with the load store control unit 42. And the increment of the numbers of words is carried out only for 1 at step 327, and processing progresses to step 330. If it is judged that numbers of words are even in step 320, processing will progress to the direct step 330.

[0070]

In step 330, it is judged whether numbers of words are larger than zero. When that is not right, it is considered that the instruction was completed and it slips out of processing at step 340. However, processing is received and passed to the register number with which progressed to step 332 and the double precision data value (namely, the contents of two data slots) was first instructed to be when numbers of words are larger than zero. Then, in step 334, the decrement of the numbers of words is carried out only for 2, and the increment of the register number is carried out only for 1 in step 336. Since a register actually consists of two data slots in a double precision instruction as stated above, it is as the same as only 2 increments the number of data slots that only 1 increments a register count.

[0071]

And it judges whether numbers of words of processing are still larger than zero to step 330 return and there. Processing is repeated when still large. If numbers of words become zero, at step 340, it will escape from processing and will come out.

[0072]

In step 310, when an instruction is judged not to be a double precision instruction, processing progresses to step 350 and it judges whether numbers of words are larger than zero again. When large, it progresses to step 352 and a single precision data value is delivered to the first register number specified with an instruction. And in step 354, only 1 carries out the decrement of the numbers of words, only 1 increments a register count in step 356, and the following data slot is directed. And it judges whether return and numbers of words of processing are still larger than zero to step 350. When large, processing is repeated until numbers of words become zero. If it becomes zero, at step 360, it will escape from processing and will come out.

[0073]

In case the above-mentioned approach performs the code which does not know the contents of the register bank, it gives big versatility for for example, a context change code, a procedure call entry, exit routine, etc. Since an operating system does not know the contents of the register in these cases, it is desirable that it is necessary to be made not to carry out treatment which changed the register with those contents. According to the above-mentioned approach, these code routines can be written in by one store or load instruction which specifies odd data word. When a co-processor needs to use the information on the contents of a register and odd data word is specified as the instruction, a co-processor is interpreted as it being the demand in which format information required since the contents of the data in a register bank are specified also loads this to memory from storing or memory. In order to support the co-processor which needs the information on the contents of a register according to this flexibility, the need of using the unique operating system software is lost.

[0074]

This approach cancels the need of loading or storing the information on the contents of a register by another operation within a code again. Since it is incorporated while the alternative which loads the information on the contents of a register, or is stored orders, additional memory access is unnecessary. Code die length becomes short by this, and time amount can be saved potentially.

[0075]

The explanation about the architecture of the floating-point unit incorporating the above-mentioned technique is shown below.

[0076]

1. Introduction VFPv1 is the architecture of the floating point system (FPS) designed so that it might realize as a co-processor used for an ARM processor module. If this architecture is carried out, the characteristic function of hardware or software is incorporable. Or the supplement of a function and IEEE754 compatibility can be offered using software. It has the intention of this specification so that IEEE754 perfect compatibility may be attained using hardware and a software support.

[0077]

Two co-processor numbers are used for VFPv1. 10 is used for the operation of a single precision operand,

and 11 is used for the operation of a double precision operand. The conversion between single precision data and double precision data is attained by two conversion command which operates in a source operand co-processor tooth space.
[0078]
The description of VFPv1 architecture is as follows.
- - which has IEEE754 and perfect compatibility using a support code in hardware Single precision register whose number is 32. Each is addressable - as a source operand or a destination register. 16 double precision registers. Each is addressable as a source operand or a destination register. (A double precision register laps with a physical single precision register)
- A vector mode raises sharply the concurrency nature of floating point code density, loading, and store actuation.
- Four banks which consist of eight circulation single precision registers, Or - to which four banks which consist of four double precision registers raise DSP and a graphic operation A non-normal processing option either IEEE754 compatibility or high-speed zero-clear capacity (to premise [support / from a floating point emulation package]) - to choose Implementation [of ****** which the pipe run was completely carried out and was connected] - which achieves results with IEEE754 compatibility A FFTOSIZ instruction is used and it is high-speed conversion from the floating point to an integer C, C++, and for Java. [0079]
An implementer can also realize VFPv1 by hardware completely, and can also use combining hardware and a support code. VFPv1 is also completely realizable with software.
[0080]
2. Terminology This specification is using the following vocabulary.
Automatic exception (automatic exception): It is in an exceptional condition. This is not concerned with the value of each exception enabling bit, but flies to a support code from this exceptional condition. Selection of which exception to make automatic is an option in a creation time point. Refer to the section 0.
[0081]
Exception handling [0082]
Bounce (bounce): It is the exception reported to the operating system, and it is completely processed in support code, without [without it calls a user trap handler, or] interrupting the normal flow of a user code.
[0083]
CDP: In 'Coprocessor Data Processing'FPS (floating point system), CDP processing is loading or not a store operation but arithmetic operation.
[0084]
ConvertToUnsignedInteger(Fm): Change the contents in Fm into a sign-less 32-bit integral value. It depends for a result on the rounding-off mode for the last rounding-off of the floating point value of a 32-bit unsigned integer out of range, and handling. A floating point input value is a negative value, or an INVALID exception may arise, when too large as a 32-bit unsigned integer.
[0085]
ConverToSignedInteger(Fm): Change the contents of Fm into a 32-bit integral value with a sign. It depends for a result on the rounding-off mode for the last rounding-off of the floating point value of a 32-bit signed integer out of range, and handling. An INVALID exception may arise, when a floating point input value is too large as a 32-bit signed integer.
[0086]
ConvertUnsignedIntToSingle/Double(Rd): Change into single precision or a double-precision-floating-point value the contents of the ARM register (Rd) decoded as a 32-bit unsigned-integer value. When destination precision is single precision, an INEXACT exception may occur by the conversion operation.
[0087]
ConvertSignedIntToSingle/Double(Rd): Change into single precision or a double-precision-floating-point value the contents of the ARM register (Rd) decoded as a 32-bit unsigned-integer value. When destination precision is single precision, an INEXACT exception may occur by the conversion operation.
[0088]
The value of the range of Denormalized value(value by which denormalization was carried out): (-2 Emin<x<2Emin) is expressed. In IEEE754 format of single precision and a double precision operand, the characteristic of a denormalization value is zero and a head bit is not 1 but 0. It is specified that 754 to IEEE1985 specification must perform creation and actuation of a denormalization operand in the same precision as a normalization operand.
[0089]

Disabled exception(forbidden exception): When the exception enabling bit to which it relates in FPCSR is set as 0, an exception says, 'It is forbidden'. IEEE754 specification defines the right result which should be returned to these exceptions. The operation which generates exception condition produces the result which carried out the bounce to the support code and was defined by IEEE754. An exception is not reported to a user exception handler.
[0090]
The exception with which the exception enabling bit Enabled exception (permitted exception) : related was set as 1. When this exception occurs, a trap is hung on a user handler. The operation which generates exception condition produces the result which carried out the bounce to the support code and was defined by IEEE754. An exception is reported to a user exception handler.
[0091]
Exponent (characteristic): In order to decide the value of the expressed figure, it is the component showing the number of integer BEKI when carrying out the exponentiation of 2 of a floating point number. A characteristic is occasionally called a sign and a sign-less characteristic.
[0092]
The field of significand (significant figure) in the right-hand side of the binary point Fraction(ed) : (decimal) suggested [0093]
Flush-To-Zero Mode: In this mode, not all the values that are in the range of (-2 Emin<x<2Emin) after rounding off are changed into a denormalization value, but are treated as zero.
[0094]
High (Fn/Fm): 32 bits [63:32] of high orders of the double precision value expressed within memory [0095]
IEEE754-1985:"IEEE Standard for Binary Floating-Point Arithmetic" and ANSI/IEEE Std 754-1985, The Institute of Electrical and Electronics Engineers, Inc.New York, New York, and 10017., although often called IEEE754 criterion This criterion is dealt with with the data type in a floating point system, a right operation, and an exception type, and defines an error limit. Almost all processors are made so that it may be based on this criterion in the combination of hardware and software in hardware.
[0096]
Infinity infinity is expressed with a special format of Infinity:IEEE754. Since all (significant figure) of precision and significand are zero, a characteristic serves as max.
[0097]
Input exception: Exception condition by which one or more operands for the given operation are not supported by hardware. In order to complete an operation, the bounce of the operation is carried out to a support code.
[0098]
Intermediate result (intermediate result): The internal format used since a count result is stored before rounding off. This format may have a bigger exponent field than a destination format and the significand (significant figure) field.
[0099]
Low (Fn/Fm): 32 bits [31:0] of low order of the double precision value expressed within memory [0100]
MCR: It is migration" to the co-processor from "ARM register. In FPS, the instruction which transmits data or a control register between an ARM register and a FPS register is included in this. Only 32-bit information can be transmitted using one MCR class instruction.
[0101]
MRC: It is migration" to the ARM register from "co-processor. In FPS, the instruction which transmits data or a control register between FPS and an ARM register is included in this. Only 32-bit information can be transmitted using one MRC class instruction.
[0102]
NaN: Not a figure but the notation-stereo coded by the floating point format. : to which two types of NaN are -- a signal and a non-signal, or quiescence. Signal NaN produces an Invalid Operand (invalid operand) exception, when using it as an operand. Quiescence NaN is spread to almost all arithmetic operation, without telling an exception. A format of NaN is significand (significant figure) all whose exponent fields are not the zero of 1. In order to express Signal NaN, as for Quiescence NaN, the most significant bit is set as 1 for the most significant bit of a decimal by zero.
[0103]
Reserved(reserved): -- if the field of a control register or an instruction format becomes "reserved" and the contents of the field are not zero, when the implementation is to define the field -- UNPREDICTABLE

(**********) -- a result is produced. These fields are reserved in order to use it at the time of a future architecture escape, or they are used only for a specific implementation. All the reserved bits that are not used by the implementation must be written in with zero, and are read as zero.
[0104]
Rounding Mode(rounding-off mode): IEEE754 specification is demanded as performing as if all count had the precision of infinity. That is, the multiplication of two single precision values must calculate significand (figure of merit) with the number of bits twice the precision of significand (figure of merit). In order to express this value with the precision of the destination, the need of rounding off significand (significant figure) often comes out. IEEE754 criterion is in :(RM) mode in which four rounding-off modes are specified. [which is rounded off to (RN) mode to round off and zero at rounding-off or (RZ) mode to omit, (RP) mode rounded off at a positive infinity, and a negative infinity] The first mode is rounded off by the midpoint, if the least significant bit of significand (figure of merit) becomes zero at the time of a tie case, it will be revalued, and it makes a figure "even number." The 2nd mode always omits substantially the bit on the right-hand side of significand (figure of merit). This is used in C, C++, and Java language in integer conversion. The two next modes are used in interval arithmetic.
[0105]
Significand (significant figure): It is the component of a binary floating point number, and consists of a head bit in the left-hand side of the suggested binary point specified or suggested, and the decimal field in right-hand side.
[0106]
Support Code(support code): It is the software which must be used in case hardware is filled up, in order to give compatibility with IEEE754 criterion. A support code has two components. One is the assembly of a routine and it performs functions currently supported, such as a division using the input which the operation beyond the range of hardware, such as transcendency count, may be performed [input], and may generate an exception and which is not supported. Another is the set of an exception handler, and it processes exception condition so that it may be based on IEEE754. A support code must perform the created function and must emulate the suitable handling of the data type which is not supported or data representation (for example, a non-legal value or a decimal data type). If attention is paid so that a user's condition may be recovered at the outlet of a routine, a routine can also be written to use FPS in middle count.
[0107]
Trap (trap): It is the exception condition which sets each exception enabling bit to FPSCR. A user's trap handler is performed.
[0108]
UNDEFINED (undefined): It is the thing of an instruction which generates an undefined-instruction trap. Refer to ARM Architectural Reference Manual about the detailed information about an ARM exception.
[0109]
It is the result of an instruction or control register field value which cannot be UNPREDICTABLE(ed) : (**********) trusted. It seems that an instruction or a result do not become a fault on secrecy, or any parts of a processor or a system are not stopped. [**********]
[0110]
Unsupported Data (data which are not supported): The specific data value which a bounce is carried out to a support code and processed without being processed by hardware. There are infinity, NaN, a non-legal value, and zero in these data. It can choose freely which is supported completely partially by hardware among these values, or whether in order to complete data processing, the assistance of a support code is needed by the implementation. If the corresponding exception enabling bit is set, the trap of the exception produced from having processed the data which are not supported will be carried out to a user code.
[0111]
3. Register file 3.1 Introduction Architecture can be equipped with 32 single precision registers and 16 double precision registers, and all registers can be addressed according to an individual within the 5-bit register index defined completely as the source or a destination operand.
[0112]
32 single precision registers have lapped with 16 double precision registers, namely, if double precision data are written in D5, they will overwrite the contents of S10 and S11. A compiler or an assembly language programmer must know that using it as one half of using a register as the single precision region of data storage and the double precision region of data storage will compete, when using a register by the overlapping implementations. Since the hardware for restricting use of a register to one precision is not

prepared, a result becomes ********** if this is not followed.
[0113]
VFPv1 enables it to access these registers by the Scala mode or the vector mode. The result produced using one, two, or three operand registers in the Scala mode is written in a destination register. Moreover, refer to the group of a register for the specified operand in a vector mode. In the case of a single precision operand, in the case of a double precision operand, VFPv1 is supporting vector operation to a maximum of four elements as opposed to a maximum of eight elements in one instruction.
表1 ＬＥＮビットコード化

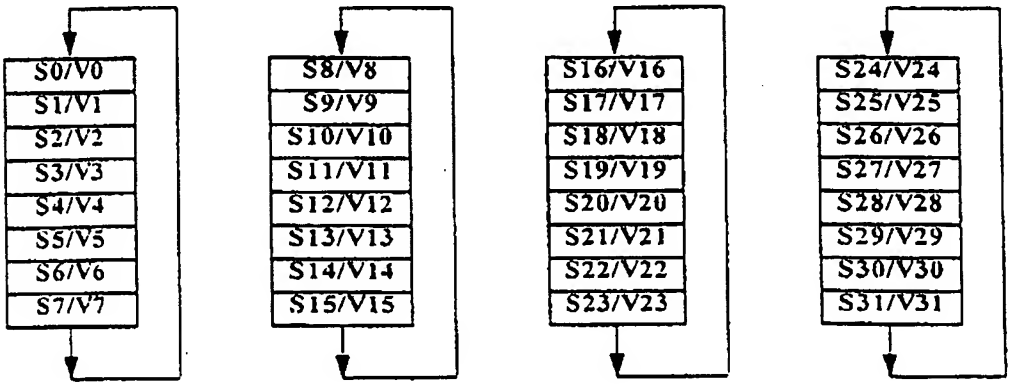
LEN	ベクトル長コード化
000	スカラ
001	ベクトル長２
010	ベクトル長３
011	ベクトル長４
100	ベクトル長５
101	ベクトル長６
110	ベクトル長７
111	ベクトル長８

[0114]
A vector mode is permitted by writing values other than zero in the LEN field. If 0 is contained in the LEN field, FPS will operate in the Scala mode and the register field will be decoded with addressing 32 single precision registers or 16 double precision registers in a flat register model. When the LEN field is not zero, FPS operates by the vector mode and the register field operates as an address vector of a register. Refer to Table 1 about the code of the LEN field.
[0115]
The approach of making Scala and vector operation intermingled can be performed by specifying a destination register, without changing the LEN field. Scalar operation can be specified in a vector mode, if a destination register is in the 1st register bank (S0-S7 or D0-D3). For details, refer to the section 0.
[0116]
3.2 Operation of single precision register When the LEN field in FPSCR is 0, 32 single precision register S0-S31 can be used. Either of these registers can be used as the source or a destination register.

31	31	31	31 0
S0	S8	S16	S24
S1	S9	S17	S25
S2	S10	S18	S26
S3	S11	S19	S27
S4	S12	S20	S28
S5	S13	S21	S29
S6	S14	S22	S30
S7	S15	S23	S31

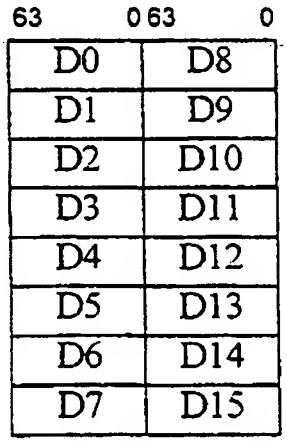
図例１　単精度レジスタマップ

A single precision (co-processor 10) register map can be created as shown in example of drawing 1.
[0117]
When the LEN field in FPSCR is larger than 0, as shown in drawing 2 , each carries out behavior of the register file as four banks which consist of eight circulating registers. the 1st of a vector register -- bank V0-V7 overlap Scala register S0-S7, and they are accessed in the address as Scala or a vector with the register chosen to each operand. Refer to the usage of a section 0 and 3.4 registers for details.



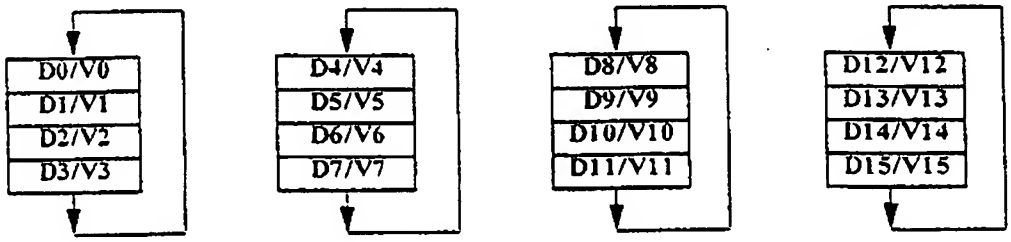
図例２　循環単精度レジスタ

[0118]
For example, if LEN in FPSCR is set to 3, and a vector V10 is referred to, registers S10, S11, S12, and S13 will be concerned with vector operation. Similarly, if V22 is referred to, S22, S23, S16, and S17 will be concerned with an operation. When a register file is accessed by the vector mode, the register following V7 is V0. In V16, V24 follows [V8] V31 following V23 following V15 back similarly.
[0119]
3.3 Usage of double precision register When the LEN field in FPSCR is 0, 16 double precision Scala registers can be used.



図例３　倍精度レジスタマップ

Any register can be used as the source or a destination register. A register map can be created as shown in example of drawing 3.
[0120]
When the LEN field in FPSCR is larger than 0, four Scala registers and 16 vector registers can use it with four bank gestalten which each becomes from four circulating registers, as shown in drawing 4 . Bank V0-V3 of the beginning of a vector register overlap Scala register D0-D3. The address of the register is carried out according to the register which the address was carried out as Scala or was chosen to each operand. Refer to a section 0 and the 3.4 register usage for details.



図例４　循環倍精度レジスタ

It circulates through a double precision register in four banks like the example of the single precision of a section 0.
[0121]
3.4 Register usage Three operations are supported between Scala and a vector. (Any of three operand operations are sufficient as;OP3 as which any of two operand operations by which OP2 is supported by the floating-point coprocessor are sufficient.) [0122]
In the case of single precision operation, in the case of the double precision operation as register S0-S7, in the following explanation, the 'first bank' of a register file is defined as register D0-D3.
- ScalarD = OP2 ScalarA or ScalarD = ScalarA OP3 ScalarB or ScalarD = ScalarA * ScalarB + ScalarD-VectorD = OP2 ScalarA or VectorD = ScalarA OP3 VectorB or VectorD = ScalarA * VectorB+VectorD-VectorD = OP2 VectorA or VectorD = VectorA OP3 VectorB or VectorD = VectorA* VectorB+VectorD
[0123]
3.4.1 Scalar operation FPS operates in the Scala mode according to two conditions.
[0124]
1? The LEN field in FPSCR is 0. In the case of single precision operation, any of the Scala register 0-31 are sufficient as the destination and a source register, and, in the case of double precision operation, any of 0-15

sufficient as them. An operation is performed only to the register specified clearly with an instruction.
[0125]
2? A destination register is in a bank of the beginning of a register file. Any of other registers are sufficient as source Scala. This mode makes it possible to make scalar operation and vector operation intermingled, without changing the LEN field in FPSCR.
[0126]
3.4.2 Operation with the vector transfer point using Scala and vector transfer origin The operation in this mode has the LEN field larger than zero in FPSCR, and when there is no destination register in a bank of the beginning of a register file, it is performed. Which register of a bank of the beginning of a register file is sufficient as a source Scala register, and each remaining register can be used for VectorB. ********** [a source Scala register is the member of VectorB, or / the behavior] when VectorD overlaps VectorB by die length shorter than a LEN element. that is, the vector with same VectorD and VectorB -- or in all members, it must be distinguished completely. Refer to the summary table of a section 0.
[0127]
3.4.3 Operation only using vector data The operation in this mode has the LEN field larger than zero in FPSCR, and when there is no destination vector register in a bank of the beginning of a register file, it is performed. Each element of a VectorA vector is combined with the corresponding element in VectorB, and is written in VectorD. Each register which is not into a bank of the beginning of a register file can use VectorA, and all vectors can be used for VectorB. ********** [the behavior] as well as the 2nd case when a source vector or a destination vector overlaps by die length shorter than a LEN element. They must be distinguished [in / same or / all members] completely. Refer to the summary table of a section 0.
In the case of the operation of a FMAC family, a destination register or a vector is always a accumulation register or a vector.
[0128]
3.4.4 Operation summary table The following tables show the option about the register use to **, double precision 2, and 3 operand instructions. It is shown that all registers can be used in the precision over the operand specified as 'Any (any)'.

表2 単精度3オペランドレジスタ使用法

LEN フィールド	転送先 Reg	第1転送元 Reg	第2転送元 Reg	演算タイプ
0	どれでも	どれでも	どれでも	S = S op S or S = S * S + S
0でない	0-7	どれでも	どれでも	S = S op S or S = S * S + S
0でない	8-31	0-7	どれでも	V = S op V or V = S * V + V
0でない	8-31	8-31	どれでも	V = V op V or V = V * V + V

表3 単精度2オペランドレジスタ使用法

LEN フィールド	転送先 Reg	転送元 Reg	演算タイプ
0	どれでも	どれでも	S = op S
0でない	0-7	どれでも	S = op S
0でない	8-31	0-7	V = op S
0でない	8-31	8-31	V = op V

表4 倍精度3オペランドレジスタ使用法

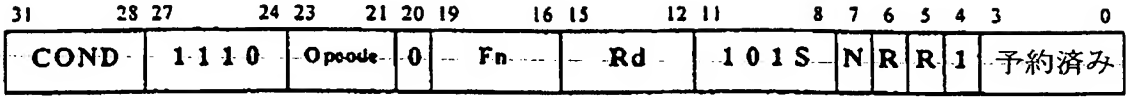
LEN フィールド	転送先 Reg	第1転送元 Reg	第2転送元 Reg	演算タイプ
0	どれでも	どれでも	どれでも	S = S op S or S = S * S + S
0でない	0-3	どれでも	どれでも	S = S op S or S = S * S + S
0でない	4-15	0-3	どれでも	V = S op V or V = S * V + V
0でない	4-15	4-15	どれでも	V = V op V or V = V * V + V

表5 倍精度2オペランドレジスタ使用法

LEN フィールド	転送先 Reg	転送元 Reg	演算タイプ
0	どれでも	どれでも	S = op S
0でない	0-3	どれでも	S = op S
0でない	4-15	0-3	V = op S
0でない	4-15	4-15	V = op V

[0129]

4. Instruction set A FPS instruction can be divided into three categories.
- Transfer [between MCR, MRC:ARM, and FPS] - Loading and store - between LDC, STC:FPS, and memory CDP: Data processing [0130]
4.1 Instruction synchronia Two level is shown in the intention of a FPS architecture specification. A pipeline functional unit and concurrency load / store actuation which has a CDP function. In case it performs in parallel with the actuation under current processing, when these actuation supports loading and store actuation without a register dependency, the big improvement in the engine performance is obtained.
[0131]
4.2 Serial-izing of instruction By FPS, ARM has specified the single instruction which keeps FPS waiting until all instructions under current activation are completed, and until each exception condition is known. If the exception is undecided, a serial-ized instruction will be stopped and exception handling will be started in ARM. The serial-ized instruction in FPS is :-. FMOVX: Read or write in to a floating-point-system register.
[0132]
The read in or the writing to a floating-point-system register is interrupted until a current instruction is completed. FMOVX to System ID Register (FPSID) is started by the exception generated by the floating point instruction to precede. The read in / modification / writing to User Status and Control Register (FPSCR) (using FMOVX) can be performed, and an exception-condition bit (FPSCR [4:0]) can be cleared.
[0133]
4.3 Conversion using integer data The conversion between the floating point and an integer data is 2 step process in FPS. That is, it consists of data transfer instruction treating an integer data, and a CDP instruction which performs conversion. When arithmetic operation is tried to the integer data of a FPS register with an integer format, and such an operation must be avoided. [the result]
[0134]
4.3.1 Conversion to floating point data from integer data of FPS register An integer data is MCR. It can load to a floating point single precision register from an ARM register using a FMOVS instruction. And the integer data in a FPS register is changed into single precision or a double-precision-floating-point value, and is written in a destination FPS register by a series of integer / floating point conversion operations. A destination register may be a source register when an integral value is unnecessary. An integer can be made into a sign and the amount of sign-less 32 bits.
[0135]
4.3.2 Conversion to integer data from floating point data of FPS register The value of FPS single precision or a double precision register is convertible for sign[a sign and]-less 32 binary-integer format with a series of floating point / integer conversion operations. The acquired integer is put in by the destination single precision register. An integer data is MRC. It is storable in an ARM register using a FMOVS instruction.
[0136]
4.4 Access by the address of register file The instruction which operates in a single precision tooth space (S= 0) uses 5 bits of an instruction field for operand access. 4 bits of high orders are contained in the operand field by which the label was carried out to Fn, Fm, or Fd. The least significant bit of the address is contained in N, M, or D.
[0137]
The instruction which operates in a double precision tooth space (S= 1) uses only 4 bits of high orders of the operand address. These 4 bits are contained in Fn, Fm, and Fd field. N, M, and D bit must be 0 when the operand address is contained in the corresponding operand field.
[0138]
4.5 MCR (it Moves to Co-processor from ARM Register)
MCR actuation is transmitting or using the data in an ARM register by FPS. This includes the actuation which is made to move data to a FPS register from the ARM register of a pair in a double precision format, loads a sign and a unsigned-integer value to a single precision FPS register from an ARM register, and loads the contents of the ARM register to a control register further from an ARM register in a single precision format.
A format of an MCR instruction is shown in example of drawing 5.



図例5 MCR命令フォーマット

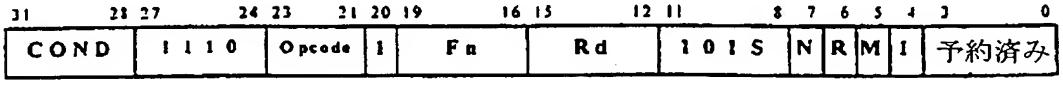
表6 MCRビットフィールドの定義

ビットフィールド	定義
Opcode	3ビット演算コード(表7参照)
Rd	ARM転送元レジスタコード
S	演算オペランドサイズ 0： 単精度オペランド 1： 倍精度オペランド
N	単精度演算： 転送先レジスタの最下位ビット 倍精度演算： 0に設定しなければならない。さもなければ演算は未定義。 システムレジスタは以下を移動する 予約済み
Fn	単精度演算： 転送先レジスタアドレス上位4ビット 倍精度演算： 転送先レジスタアドレス システムレジスタは以下を移動する： 0000-FPID（コプロセッサID番号） 0001-FPSCR(ユーザステータスおよび制御レジスタ) 0100-FPREG（レジスタファイル内容レジスタ） 他のレジスタのコードは予約済みであり、インプリメンテーションによって異なることがある。
R	予約ビット

表７　ＭＣＲ演算コードフィールド定義

Opcode フィールド	名称	動作
０００	ＦＭＯＶＳ	F=Rd（３２ビット、コプロセッサ１０）
０００	ＦＭＯＶＬＤ	Low(Fn)=Rd(倍精度下位３２ビット、コプロセッサ１１）
００１	ＦＭＯＶＨＤ	High(Fn)=Rd(倍精度上位３２ビット、コプロセッサ１１）
０１０－１１０	予約済み	
１１１	ＦＭＯＶＸ	System Reg=Rd （コプロセッサ１０スペース）

[0139]
Notes: Only 32 bit-data processing is supported with a FMOV [S, HD, LD] instruction. Only the data of an ARM register or a single precision register are moved by FMOVS actuation. In case a double precision operand is transmitted from two ARM registers, FMOVLD and a FMOVHD instruction move in a lower half and an upper half, respectively.
[0140]
4.6 MRC (it Moves to ARM Register from Co-processor / Comparison Floating-point Register)
MRC actuation transmits the data of a FPS register to an ARM register. This moves migration or a double precision FPS register for the result of having changed the single precision value or the floating point value into the integer to an ARM register to two ARM registers, and includes the actuation which changes the status bit of CPSR by the result of a pre- floating-point-compare operation.
A format of a MRC instruction is shown in example of drawing 6.



図例6　MRC命令フォーマット

表８　ＭＲＣビットフィールド定義

ビットフィールド	定義
Opcode	3ビットＦＰＳ演算コード(表９参照)
Rd	ＡＲＭ転送元＊レジスタコード
S	演算オペランドサイズ ０：　単精度オペランド １：　倍精度オペランド
N	単精度演算： 転送先レジスタの最下位ビット 倍精度演算： ０に設定しなければならない。さもなければ演算は未定義。 システムレジスタは以下を移動する 予約済み
M	予約済み
Fn	単精度演算： 転送先レジスタアドレス上位4ビット 倍精度演算： 転送先レジスタアドレス システムレジスタは以下を移動する： ００００-ＦＰＩＤ（コプロセッサＩＤ番号） ０００１-ＦＰＳＣＲ(ユーザステータスおよび制御レジスタ) ０１００-ＦＰＲＥＧ（レジスタファイル内容レジスタ) 他のレジスタのコードは予約済みであり、インプリメンテーションによって異なることがある。
Fm	予約ビット
R	予約ビット

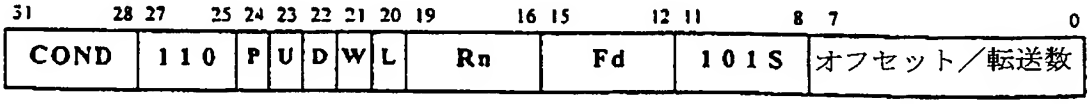
＊ＦＭＯＶＸ　ＦＰＳＣＲ命令の場合、もしＲｄフィールドにＲ１５(１１１１)が入っているなら、ＣＰＳＲの上位４ビットは得られた条件コードで更新される。

表９　ＭＲＣ演算コードフィールド定義

Opcode フィールド	名称	動作
０００	ＦＭＯＶＳ	Rd=Fn（３２ビット、コプロセッサ１０）
０００	ＦＭＯＶＬＤ	Rd=Low(Fn)Dnの下位32ビッドが転送される。（倍精度下位３２ビット、コプロセッサ１１）
００１	ＦＭＯＶＨＤ	Rd=High(Fn)Dnの上位３２ビットが転送される。（倍精度上位３２ビット、コプロセッサ１１）
０１０－１１０	予約済み	
１１１	FMOVX	Rd=System Reg

注：ＭＣＲ　ＦＭＯＶ命令の注記を参照。

[0141]
4.7 LCD/STC (loading/store FPS register) LDC and STC actuation transmit data between FPS and memory. A floating point data updates an ARM address register, or is in a condition as it is, and can transmit it in any precision by single data transfer or two or more data transfer. The structure of both a full descending-order stack and an empty ascending-order stack is supported. Furthermore, two or more operands access to DS is also supported with two or more migration instruction. Refer to Table 11 about explanation of the various options of LDC and STC.
A format of LDC and an STC instruction is shown in example of drawing 7.



図例7　LDC/STC命令フォーマット

表１０　ＬＤＣ／ＳＴＣビットフィールド定義

ビットフィールド	定義
P	前／後インデクシング（０＝後、１=前）
U	上／下ビット（０＝下、１＝上）
D	単精度演算： 転送元/転送先レジスタ最下位ビット 倍精度演算： ０に設定しなければならない
W	ライトバックビット（０=ライトバックなし、１=ライトバック）
L	方向ビット（０=ストア、１=ロード）
Rn	ARMベースレジスタコード
Fd	単精度演算： 転送元/転送先レジスタアドレスでアクセス上位４ビット 倍精度演算： 転送元/転送先レジスタアドレス
S	演算オペランドサイズ ０：単精度オペランド １：倍精度オペランド
ｵﾌｾｯﾄ/転送数	FLDM(IA/DB)およびFSTM(IA/DB)に対して転送すべき符号なし８ビットオフセットまたは単精度レジスタ数（倍精度レジスタ数の２倍）。１回の転送の最大のワード数は１６。これで１６個の単精度値または８個の倍精度値の転送が可能。

[0142]
4.7.1 General-precautions point about loading and store actuation Loading and a store of two or more registers are linearly performed through a register file without carrying out a surroundings lump in the range which consists of 4 or eight registers which are used by vector operation. ********** [the result] when loading is tried exceeding the termination of a register file.
[0143]
Although 32 still more nearly another bit-data items can be written in or 32 another bit-data items can be read in an implementation when 17 or the number of registers of the odd number not more than it is contained in offset of a double load or two or more stores, it is not necessary to necessarily make it such. Those contents can be specified, in case this additional data item is used and a register is loaded or stored. This is helpful when a register file format has the type information which needs each register for the discernment in memory of itself unlike IEEE754 format of the precision. When offset is larger than the number of single precision registers with odd number, it can use for this starting the context change of a register, and all system registers.

表１１　ロードおよびストアアドレス指定モードオプション

P	W	オフセット／転送数	アドレス指定モード	名称
タイプ０転送：ライトバックなしに複数をロード/ストアする				
0	0	転送すべきレジスタの数	FLDM<cond><S/D>Rn.<レジスタリスト> FSTM<cond><S/D>Rn.<レジスタリスト>	ロード/ストアマルチプル
Rnにおける先頭アドレスから複数のレジスタをロード/ストア。Rnの変更なし。レジスタの数は単精度の場合１-１６個、倍精度の場合１-８個。オフセットフィールドには３２ビット転送の数が入っている。このモードを使ってグラフィックス演算用の変換マトリックスおよびその変換点をロードすることができる。				
例： FLDMEQSr12,{f8-f11}；ｒ１２のアドレスから４個の単精度を４個のｆｐレジスタｓ８、ｓ９、ｓ１０へロードする。ｒ１２は不変。 FSTMEQDr4,{f0}；ｄ０から１個の倍精度をｒ４のアドレスへストアする。ｒ４は不変。				
タイプ１転送：複数をロード/ストアする。Rnのポストインデックスとライトバックあり。				
0	1	転送すべきレジスタの数	FLDM<cond>IA<S/D>Rn!.<レジスタリスト> FSTM<cond>IA<S/D>Rn!.<レジスタリスト>	ロード/ストアマルチプル
Rnにおける先頭アドレスから複数のレジスタをロード/ストアし、最後の転送の後に次のアドレスをRnへライトバック。オフセットフィールドには３２ビット転送の数が入っている。Rnへのライトバックはオフセット＊４。ロードマルチプルにおいて転送された最大ワード数は１６。Uビットは１にセットしなければならない。これは空の昇順スタックへ格納するために、あるいは満杯の降順スタックからロードするために使用する。あるいは変換点を格納し、次の点へポインタをインクリメントするために、またフィルタ動作において複数データをロード/ストアするために使用する。				
例： FLDMEQIASr13!,{f12-f15}；ｒ１３のアドレスカラ４個の単精度を４個のｆｐレジスタｓ１２、ｓ１３、ｓ１４、ｓ１５へロードする。ｒ１３は次のデータを指すアドレスで更新する。				
タイプ２転送：１個のレジスタをロード/ストアする。Rnのプリインデックスあり、ライトバックなし。				

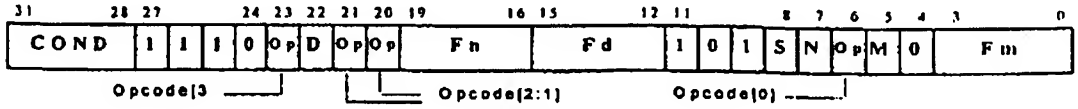
1	0	オフセット	FLD<cond><S/D>[Rn.#+/-offset], Fd FST<cond><S/D>[Rn.#+/-offset], Fd	オフセットありロード/ストア
１個のレジスタをロード/ストアする。Rnのアドレスのプリインクリメントあり、ライトバックなし。オフセット値はオフセット＊４であり、Rnに加える（U=1）かそれから差し引く（U=0）ことによりアドレスを発生する。これは、オペランドアクセスのために有用であり、浮動小数点データを取り出すためにメモリをアクセスする典型的な方法である。				
例： FSTEQDf4,[r8,#+8]； ３２（８＊４）バイトオフセットしたｒ８のアドレスカラ１個の倍精度をｄ４へ格納する。ｒ８は不変。				
タイプ３転送：複数レジスタをロード/ストアする。プリインデックスとライトバックあり。				
1	1	転送すべきレジスタの数	FLDM<cond>DB<S/D>Rn!.<レジスタリスト> FSTM<cond>DB<S/D>Rn!.<レジスタリスト>	プリデクリメント有りロード/ストアマルチプル
複数のレジスタをロード/ストアする。Rnのアドレスをプリデクリメントし、新しいターゲットアドレスをRnにライトバック。オフセットフィールドには３２ビット転送の数が入っている。ライトバック値はオフセット＊４であり、Rnから差し引く（U=0）。このモードは満杯の降順スタックへストアするか、空の昇順スタックからロードするのに使用する。				
例： FSTMEQDBSr9!,{f27-f29}； ｓ２７、ｓ２８、ｓ２９から３個の単精度を満杯の降順スタックへストアする。最後のエントリアドレスはｒ９に入っている。ｒ９は新しい最後のエントリを指すように更新する。				

[0144]
4.7.2 LDC/STC actuation summary P [in / in Table 12 / LDC/STC operation code], W, and the combination of U bits permitted -- each -- the function of offset to an effective operation is shown.

表１２　ＬＤＣ／ＳＴＣ動作サマリ

P	W	U	オフセットフィールド	動作
0	0	0		未定義
0	0	1	レジスタカウント	FLDM/FSTM
0	1	0		未定義
0	1	1	レジスタカウント	FLDMIA/FSTMIA
1	0	0	オフセット	FLD/FST
1	0	1	オフセット	FLD/FST
1	1	0	レジスタカウント	FLDMDB/FSTMDB
1	1	1		未定義

[0145]
4.8 CDP (co-processor data processing) A CDP instruction produces a result using the operand from a floating point register file, and includes all data-processing actuation of the result in which it is returned to a register file. It is a FMAC (multiplication-accumulation was connected) operation that there is especially interest, and this is an operation which multiplies two operands and adds the 3rd operand. Before an IEEE rounding-off operation adds the 3rd operand, this operation is the point performed to a product, and differs from a fusion ****** operation. Thereby, the Java code can speed up [of a ****** operation] by using a FMAC operation rather than the operation added after multiplication.
[0146]
Two instructions in a CDP group are helpful when changing the floating point value in a FPS register into the integral value. FFTOUI [S/D] changes the contents of single precision or the double precision register into the unsigned integer in a FPS register using the present rounding-off mode in FPSCR. FFTOSI [S/D] performs conversion to a signed integer. Although FFTOUIZ [S/D] and FFTOSIZ [S/D] perform the same function, they make an invalid FPSCR rounding-off mode for conversion, and omit a decimal bit. In case the function of FFTOSIZ [S/D] is changed into an integer from the floating point, it is required by C, C++, and Java. Since a FFTOSIZ [S/D] instruction offers this capacity, without adjusting the rounding-off mode bit for conversion from FPSCR to RZ, it decreases even to the cycle count of a FFTOSIZ [S/D] operation, and it carries out 4-6 cycle saving of the cycle count for conversion.
[0147]
A comparison operation is CDP. MRC following a CMP instruction and it FMOVX It is ARM about the FPS flag bit (FPSCR [31:28]) which were obtained by carrying out using the FPSCR instruction. It loads to a CPSR flag bit. A compare instruction does not have the possibility of an INVALID (invalid) exception, when one of the comparison operands is NaN. Although FCMP and FCMP0 do not tell INVALID (invalid) when one of the comparison operands is NaN, FCMPE and FCMPE0 tell an exception. FCMP0 and FCMPE0 compare the operand in Fm field with 0, and they set a FPS flag according to the result. The ARM flags N, Z, C, and V are FMOVX. It defines as follows after a FPSCR instruction.
N: There are also few twists. Z: It is equal. C: With [or it is large / be / it / equal or] no sequence V: With no sequence A format of a CDP instruction is shown in example of drawing 8.



図例8　CDP命令フォーマット

表１３　ＣＤＰビットフィールド定義

ビットフィールド	定義
Opcode	４ビットＦＰＳ演算コード（表１４参照）
D	単精度演算： 転送先レジスタ最下位ビット 倍精度演算： ０にセットしなければならない
Fn	単精度演算： 転送元Aレジスタ４ビットまたは 演算コード最上位４ビットを拡張 倍精度演算： 転送元Aレジスタアドレスまたは 演算コード最上位４ビットを拡張
Fd	単精度演算： 転送先レジスタ４ビット 倍精度演算： 転送先レジスタアドレス
S	演算オペランドサイズ ０：単精度オペランド １：倍精度オペランド
N	単精度演算： 転送元Aレジスタ最下位ビット 演算コード最下位ビットを拡張 倍精度演算： ０にセットしなければならない 演算コード最下位ビットを拡張
M	単精度演算： 転送元Bレジスタ最下位ビット 倍精度演算： ０にセットしなければならない
Fm	単精度演算： 転送元Bレジスタアドレス上位４ビット 倍精度演算： 転送元Bレジスタアドレス

[0148]
4.8.1 Operation code Table 14 shows the elementary operation code of a CDP instruction. All mnemonic codes take the form of [OPERATION], [COND], and [S/D].

表１４　ＣＤＰ演算コード仕様

演算コードフィールド	命令名称	演算
００００	FMAC	Fd=Fn*Fm+Fd
０００１	FNMAC	Fd=-(Fn*Fm+Fd)
０００１０	FMSC	Fd=Fn*Fm-Fd
０００１１	FNMSC	Fd=-(Fn*Fm-Fd)
０１００	FMUL	Fd=Fn*Fm
０１０１	FNMUL	Fd=-(Fn*Fm)
０１１０	FSUB	Fd=Fn-Fm
０１１１	FNSUB	Fd=-(Fn-Fm)
１０００	FADD	Fd=Fn+Fm
１００１-１０１１	予約済み	
１１００	FDIV	Fd=Fn/Fm
１１０１	FRDIV	Fd=Fm/Fn
１１１０	FRMD	Fd=Fn%Fm(Fd=Fn/Fmの後に残る小数)
１１１１	拡張	Fnレジスタフィールドを使って２オペランド演算のための命令を指定する(表１５を参照)

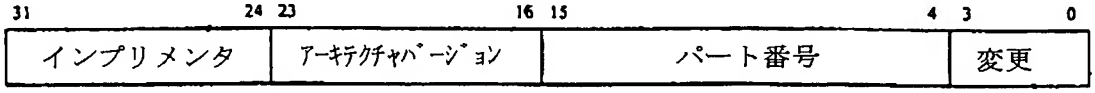
[0149]
4.8.2 Extended arithmetic Table 15 shows the extended instruction using Extend (extended value) of the operation code field. Although all instructions take the form of [OPERATION], [COND], and [S/D], serialization and a FLSCB instruction are exceptions. The instruction code of extended arithmetic is made like the index to the register file of Fn operand, i.e., {Fn [3:0], N}.

表１５　ＣＤＰ拡張命令

Fn｜N	名称	演算
００００００	FCPY	Fd=Fm
００００１	FABS	Fd=abs(Fm)
００００１０	FNEG	Fd=-(Fm)
００００１１	FSQRT	Fd=sqrt(Fm)
００１００-００１１１	予約済み	
０１０００	FCMP*	Flags:Fd⇔Fm
０１００１	FCMPE*	Flags:=Fd⇔Fm例外報告有り
０１０１０	FCMP0*	Flags:=Fd⇔0
０１０１１	FCMPE0*	Flags:=Fd⇔0例外報告有り
０１１００-０１１１０	予約済み	
０１１１１	FCVTD<cond>S*	Fd(倍精度レジスタコード)=Fm(単精度レジスタコード)単精度から倍精度に変換されたもの。（コプロセッサ１０）
０１１１１	FCVTS<cond>D*	Fd(単精度レジスタコード)=Fm(倍精度レジスタコード)倍精度から単精度に変換されたもの。（コプロセッサ１１）
１００００	FUITO*	Fd=符号なし整数を単/倍精度に変換（Fm)
１０００１	FSITO*	Fd=符号あり整数を単/倍精度に変換（Fm)
１００１０-１０１１１	予約済み	
１１０００	FFTOUI*	Fd=符号なし整数に変換（Fm)｛現行RMODE｝
１１００１	FFTOSIZ*	Fd=符号なし整数に変換（Fm)｛RZモード｝
１１０１０	FFTOSI*	Fd=符号あり整数に変換（Fm)｛現行RMODE｝
１１０１１	FFTOSIZ*	Fd=符号あり整数に変換（Fm)｛RZモード｝
１１１００-１１１１１	予約済み	

＊ベクトル化不可能な命令。LENフィールドは無視され、スカラ演算が指定されたレジスタに対して行われる。

[0150]
5. System Register 5.1 System ID Register (FPSID)
FPS architecture and an implementation definition ID value are included in FPSID.



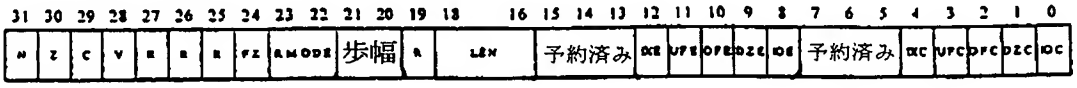
図例9　FPSIDレジスタコード

It can opt for the model of FPS and a mask set number, the description set, and modification using this WORD. FPSID is only for read in and the writing to FPSID is disregarded. Refer to example of drawing 9 about a FPSID register layout.

[0151]

5.2 User Status and Control Register (FPSCR)

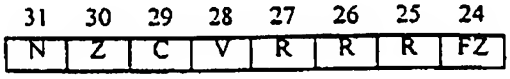
A FPSCR register contains a configuration bit with an accessible user, and an exception status bit. An exception authorization bit, rounding control, a vector step and die length, a non-normal operand, handling of a result, and use of a debug mode are included in a configuration option. An operating system uses this register with a user, FPS is constituted or it is used for asking the condition of the completed instruction. This must save and must be recovered at the time of a context change. Bits 31-28 have a flag value from the newest compare instruction, and can be accessed using the read in of FPSCR. FPSCR is shown in example of drawing 10.



図例10 ユーザステータスと制御レジスタ(FPSCR)

[0152]

5.2.1 Status comparison and processing control byte Bits 31-28 have some useful control bits, although the arithmetic response of FPS is specified in the result of the newest compare instruction, and a special situation. A format of a status comparison and a processing control byte is shown in example of drawing 11.



図例11 FPSCRステータス比較および処理制御バイト

表16 FPSCRステータス比較および処理制御バイトフィールド定義

レジスタビット	名称	機能
31	N	比較結果は…より小さい
30	Z	比較結果は…と等しい
29	C	比較結果は…より大きいまたは等しいまたは順列なし
28	V	比較結果は順列なし
27：25	予約済み	
24	FZ	ゼロにクリア 0：IEEE754アンダフロー処理(デフォルト) 1：小さい値の結果はゼロにする 転送先精度の正規範囲よりも小さい結果は、転送先レジスタにゼロを書き込む。アンダフロー例外トラップは取らない。

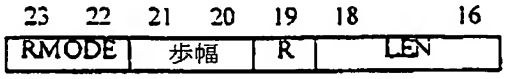
[0153]

5.2.2 System control cutting tool A system control cutting tool controls rounding-off mode, a vector step, and the vector die-length field. A bit is specified as shown in example of drawing 12.

[0154]

VFPv1 architecture has incorporated the register file step device used for vector operation. When the step bit is set to 00, the register chosen as a degree in vector operation turns into a register just behind a front register within a register file. The circumference lump device of a normal register file is not influenced depending on a step value. When a step value is 11, only 2 increments all input registers and output registers.

For example, on :FMULEQS F8 which performs the following non-vector operation, F16, F24FMULEQS F10, F18, F26FMULEQS F12, F20, F28FMULEQS F14, F22, and an F30 real target, FMULEQS F8, F16, and F24 are not one registers, and are straddling two registers of operands for multiplication in a register file at a time.



図例12 FPSCRシステム制御バイト

表１７ ＦＰＳＣＲシステム制御バイトフィールド定義

レジスタビット	名称	機能
２３：２２	RMODE	丸めモードを設定 ００：RN(四捨五入；デフォルト) ０１：RP(正の無限大に丸める) １０：RM（負の無限大に丸める） １１：RZ(ゼロに丸める)
２１：２０	歩幅	ベクトルレジスタアクセスを以下に設定： ００：１(デフォルト) ０１：予約済み １０：予約済み １１：２
１９	予約済み(R)	
１８：１６	LEN(長さ)	ベクトル長さ。ベクトル演算の長さを指定する。（すべてのコードがそれぞれのインプリメンテーションで使用できるのではない） ０００：１(デフォルト) ００１：２ ０１０：３ ０１１：４ １００：５ １０１：６ １１０：７ １１１：８

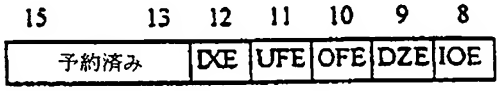
[0155]

5.2.3 Exception authorization cutting tool An exception authorization cutting tool occupies a bit 15:8, and has enabling for exception traps. A bit is specified as shown in example of drawing 13. The exception enabling bit has agreed in the demand of the IEEE754 use for processing of floating point exception

condition. If the bit is set, an exception will be permitted, and FPS tells a user visible trap to an operating system, when exception condition occurs about a current instruction. If the bit is cleared, an exception will not be permitted, and FPS does not tell a user visible trap to an operating system, even if an exception occurs. However, a mathematically rational result is generated. The default of an exception enabling bit is forbidden. Refer to IEEE754 criterion about the detail about exception handling.

[0156]

Even if the exception is a disable depending on the implementation, in order to deal with the exception condition besides the capacity of hardware, the bounce to a support code may be generated. This looks general to a user code.



図例13 FPSCR例外イネーブルバイト

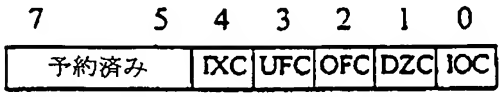
表１８ ＦＰＳＣＲ例外イネーブルバイトフィールド

レジスタビット	名称	機能
１５：１３	予約済み	
１２	IXE	不正確なイネーブルビット ０：禁止(デフォルト) １：許可
１１	UFE	アンダフローイネーブルビット ０：禁止(デフォルト) １：許可
１０	OFE	オーバフローイネーブルビット ０：禁止(デフォルト) １：許可
９	DZE	ゼロで割るイネーブルビット ０：禁止(デフォルト) １：許可
８	IOE	無効オペランドイネーブルビット ０：禁止(デフォルト) １：許可

[0157]

5.2.4 Exception status byte An exception status byte occupies the bit 7:0 of FPSCR, and has an exception status flag bit. There are five exception status flag bits and one flag bit supports each floating point exception at a time. 'It has pasted' up, and if these bits are set by the once detected exception, they must be cleared with the FMOVX instruction or FSERIALCL instruction written in FPSCR. These bits are specified as shown in example of drawing 14. When an exception is permitted, a corresponding exception status bit is not set automatically. The role of a support code sets a suitable exception status bit if needed. A certain exception may be performed automatically, namely, if exception condition is detected, FPS will not be concerned with how the exception enabling bit was set, but will carry out a bounce by the consecutive floating point instruction. Complicated exception handling can be performed not by hardware but by software rather than it is required by IEEE754 criterion by this. As an example, there are underflow conditions by which FZ bit was set to 0. In this case, a right result may be a denormalized number depending on the characteristic and rounding-off mode of a result. By FPS, an implementer can choose the response including a bounce option, can make a right result using a support code, and can write this value in

a destination register. If the underflow exception enabling bit is set, a user trap handler will be called after a support code completes an instruction. This code can change the condition of FPS, and can return, or can end processing.

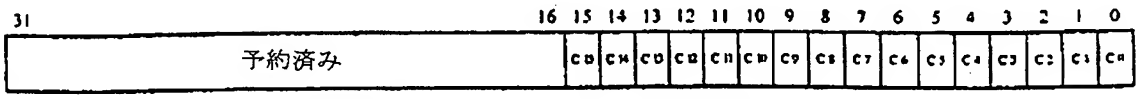


図例14 FPSCR例外ステータスバイト

表１９ ＦＰＳＣＲ例外ステータスバイトフィールド定義

レジスタビット	名称	機能
7：5	予約済み	
4	IXC	不正確な例外検出
3	UFC	アンダフロー例外検出
2	OFC	オーバフロー例外検出
1	DZC	ゼロで割る例外検出
0	IOC	無効演算例外検出

[0158]
5.3 Contents Register of Register File (FPREG)
Since the contents of a register [decoding by the program which is running now] are expressed appropriately, the contents register of a register file is a privileged register which has the information which a debugger can use. FPREG has 16 bits and is assigned 1 bit at a time to each double precision register in a register file. A set of a bit displays the physical register pair expressed with the bit as a double precision register. If a bit is cleared, a physical register is not initialized or has one or two single precision data values.



図例15 FPREGビットフィールドの定義

表２０　ＦＰＲＥＧビットフィールド定義

ＦＰＲＥＧビット	ビットセット	ビットクリア
C0	D0有効	S1とS0有効または未初期化
C1	D1有効	S3とS2有効または未初期化
C2	D2有効	Ｓ５とＳ４有効または未初期化
C3	D3有効	Ｓ７とＳ６有効または未初期化
C4	D4有効	Ｓ９とＳ８有効または未初期化
C5	D5有効	Ｓ１１とＳ１０有効または未初期化
C6	D6有効	Ｓ１３とＳ１２有効または未初期化
C7	D7有効	Ｓ１５とＳ１４有効または未初期化
C8	D8有効	Ｓ１７とＳ１６有効または未初期化
C9	D9有効	Ｓ１９とＳ１８有効または未初期化
C10	D10有効	Ｓ２１とＳ２０有効または未初期化
C11	D11有効	Ｓ２３とＳ２２有効または未初期化
C12	D12有効	Ｓ２５とＳ２４有効または未初期化
C13	D13有効	Ｓ２７とＳ２６有効または未初期化
C14	D14有効	Ｓ２９とＳ２８有効または未初期化
C15	D15有効	Ｓ３１とＳ３０有効または未初期化

[0159]
6. Exception handling FPS operates in one of the modes among two mode, debug mode, and normal-mode **s. If DM bit is set to FPSCR, FPS will operate by the debug mode. In this mode, FPS executes one instruction at once, and ARM is carried out again in the meantime until the exception condition of an instruction is known. Although a register file and memory become exact about the flow of an instruction by this, the sacrifice that the execution time increases sharply will be paid. If a resource allows FPS, when a new instruction is received from ARM and exception condition is detected, it tells an exception. Exception reporting to ARM is always exact about a floating-point-instruction train. In the case of loading merely performed to vector operation and juxtaposition following vector operation, or store instruction, it is an exception. The contents of the register file may become in this case, less exact [about store instruction / the contents of memory] about a load instruction again.
[0160]
6.1 Support code The implementation of FPS can be based on IEEE754 using hardware and a software support. About the data type and automatic exception which are not supported, a support code performs the function of the based hardware, returns a result to a destination register, and returns to a user's code. A user's trap handler is not called in that case, or flow of a user's code is not changed. For a user, it seems that only hardware processed the floating point code. Although carrying out a bounce to a support code reduces sharply the time amount which performs these processings in order to deal with these processings, in a user code, the embedded application, and the arithmetic application written well, it is usually rare for these situations to occur.
[0161]
It has the intention of a support code so that it may have two components. One of them is the assembly of a routine and it performs functions currently supported, such as a division using the input which the operation beyond the range of hardware, such as transcendency count, may be performed [input], and may generate an exception and which is not supported. Another is the set of an exception handler, and it processes an exception trap so that it may be based on IEEE754. A support code must perform the created function and must emulate the suitable handling of the data type which is not supported or data representation (for example, non-legal value). If attention is paid so that a user's condition may be recovered at the outlet of a routine, a routine can also be written to use FPS in middle count.

[0162]
6.2 Exception reporting and processing The following floating point instruction published after exception condition is detected reports the exception in a normal mode to ARM. The condition of an ARM processor, a FPS register file, and memory may not be exact about a violation instruction when an exception is taken. The instruction is emulated correctly and a support code is obtained in sufficient information to process the exception produced from the instruction.
[0163]
Using a support code, it is some which have special IEEE754 data which contain infinity, NaN, non-normal data, and zero depending on an implementation, or all instructions can also be processed. The implementation which carries out such processing is referred to as data which are not having these data supported, it carries out a bounce to a support code so that it may not look generally to a user code, puts an IEEE754 assignment result into a destination register, and returns. As for the exception produced from this actuation, anythings follow the IEEE754 exception Ruhr. If the corresponding exception enabling bit is set, the trap to a user code can also include this exception.
[0164]
The correspondence to exception condition is defined to the case where it is when [both] not enabling IEEE754 criterion with the case where it enables the exception bit in FPSCR. VFPv1 architecture has not specified the boundary between the hardware and software which are used so that it may be based suitable for IEEE754 specification.
[0165]
6.2.1 Instruction and format which are not supported FPS is not supporting the conversion from /to the instruction of decimal data, or decimal data. These instructions are demanded by IEEE754 criterion and must be offered in support code. When using decimal data, the assembly of the routine of a desired function is needed. Since FPS does not have a decimal data type, it cannot use in order to carry out the trap of the instruction which uses decimal data.
[0166]
6.2.2 Use of FMOVX when FPS serves as disable or exception The FMOVX instruction executed in a supervisor or undefined mode reads R/W, FPSID, or FPREG for FPSCR, without telling ARM about an exception (when the implementation supporting the disable option), when FPS is an exception condition or a disable.
[0167]
Although the specific example of this invention was described, probably, it will be clear that this invention is not restricted to them and that various modification and additions at within the limits of invention can be performed. For example, the description of the following dependent claims can be variously combined with the description of an independent claim, without deviating from the range of this invention.
[Brief Description of the Drawings]
[Drawing 1]
It is the schematic diagram of data processing system.
[Drawing 2]
It is the explanatory view of the floating-point unit which supports both the Scala register and a vector register.
[Drawing 3]
In single precision operation, it is the flow chart showing how it is decided any of a vector register or the Scala register the given registers are.
[Drawing 4]
In double precision operation, it is the flow chart showing how it is decided any of a vector register or the Scala register the given registers are.
[Drawing 5]
It is drawing showing the surroundings lump at the time of single precision operation within each subset which divided the register bank.
[Drawing 6]
It is drawing showing the surroundings lump at the time of double precision operation within each subset which divided the register bank.
[Drawing 7 A]
It is drawing showing the co-processor instruction which the co-processor instruction which a main processor looks at, single precision, and a double precision co-processor look at, and the co-processor

instruction which a single precision co-processor looks at, respectively.
[Drawing 7 B]
It is drawing showing the co-processor instruction which the co-processor instruction which a main processor looks at, single precision, and a double precision co-processor look at, and the co-processor instruction which a single precision co-processor looks at, respectively.
[Drawing 7 C]
It is drawing showing the co-processor instruction which the co-processor instruction which a main processor looks at, single precision, and a double precision co-processor look at, and the co-processor instruction which a single precision co-processor looks at, respectively.
[Drawing 8]
It is drawing showing the main processor which controls single precision and a double precision co-processor.
[Drawing 9]
It is drawing showing the main processor which controls a single precision co-processor.
[Drawing 10]
In order to show that the co-processor instruction was received, it is drawing showing the circuit in the single precision and the double precision co-processor which determine whether a reception signal should be returned to a main processor.
[Drawing 11]
In order to show that the co-processor instruction was received, it is drawing which determines whether a reception signal should be returned to a main processor and in which showing the circuit in a single precision co-processor.
[Drawing 12]
It is drawing showing the handling of the instruction exception of the undefined within a main processor.
[Drawing 13]
It is the block diagram showing the element of the co-processor by the desirable example of this invention.
[Drawing 14]
It is the flow chart showing actuation of the register control by the desirable example of this invention, and instruction issue logic.
[Drawing 15]
It is an example of the contents of the floating-point register by the desirable example of this invention.
[Drawing 16]
It is drawing showing the register bank in clay 1 processor.
[Drawing 17]
It is drawing showing the register bank in a multi-tie tongue processor.

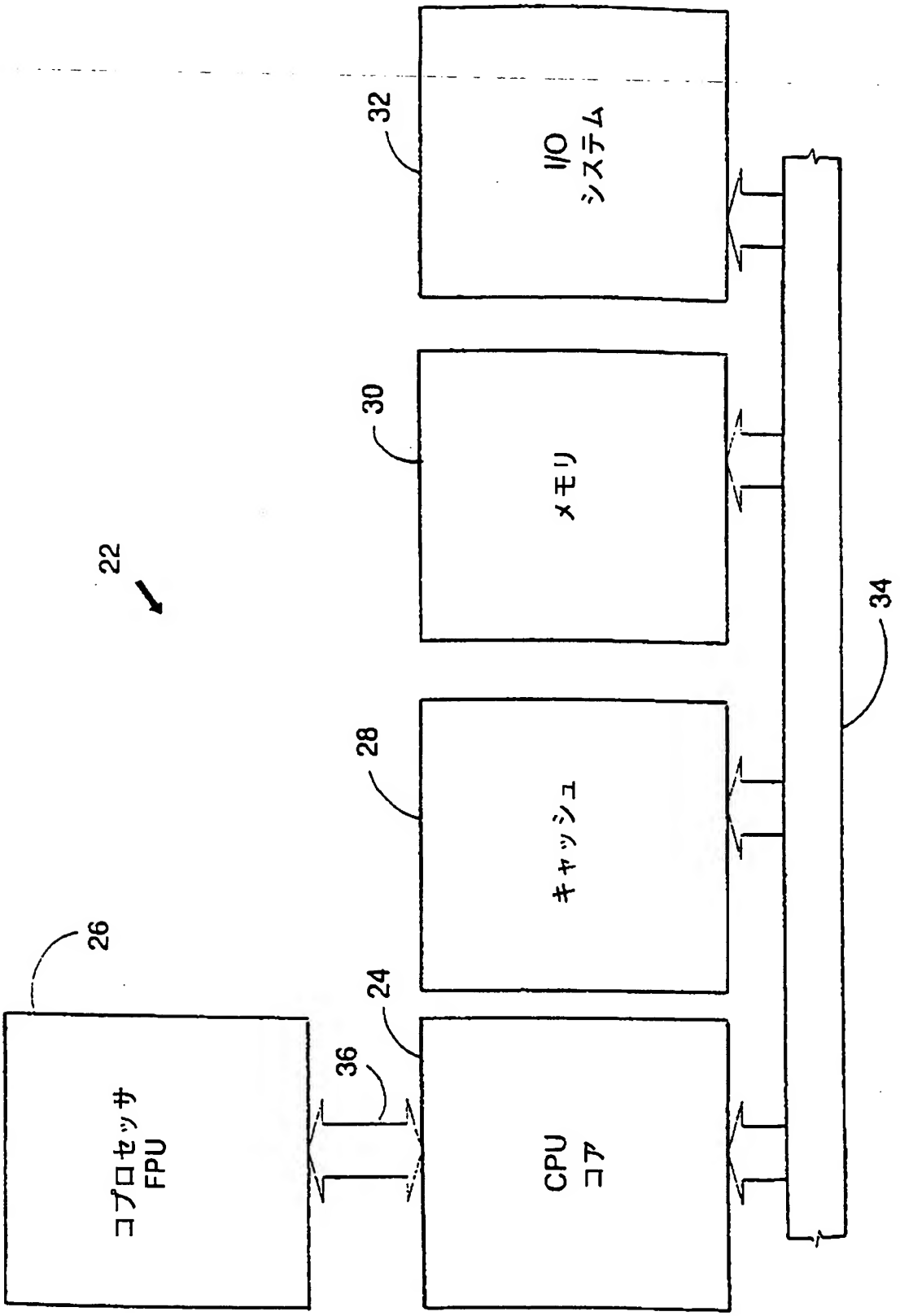
[Translation done.]

* NOTICES *

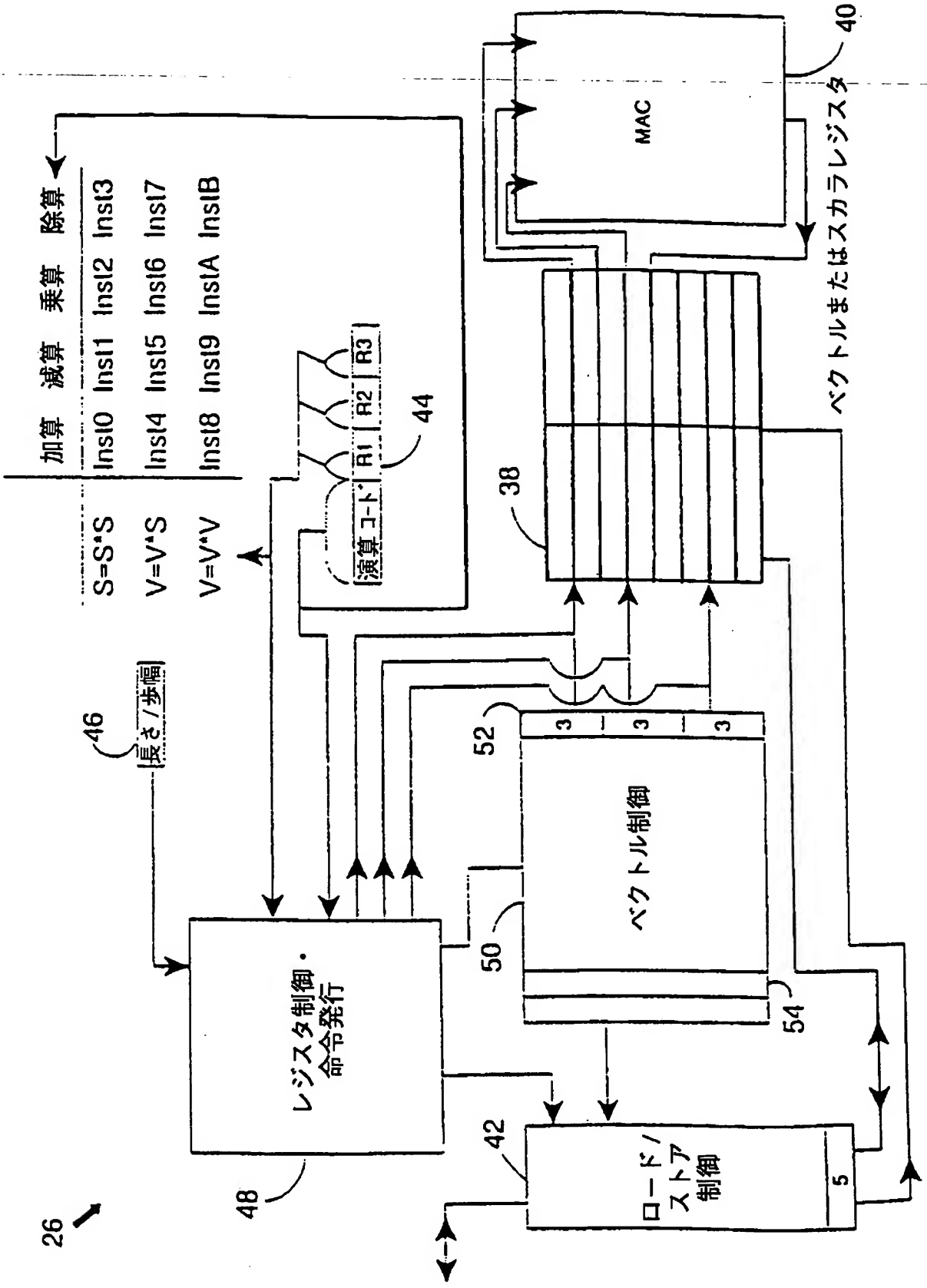
1.This document has been translated by computer. So the translation may not reflect the original precisely.
2.**** shows the word which can not be translated.
3.In the drawings, any words are not translated.

DRAWINGS

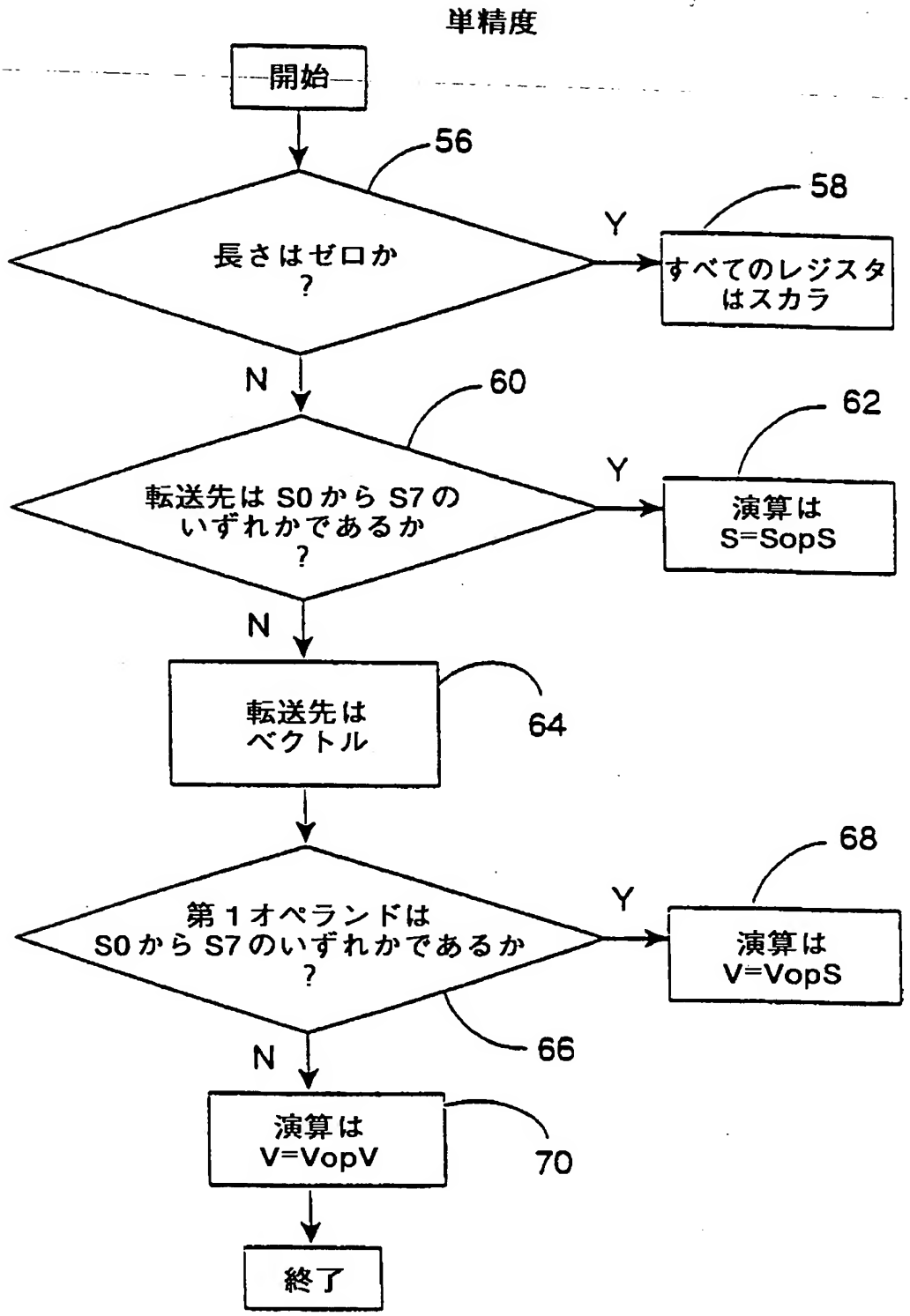
[Drawing 1]



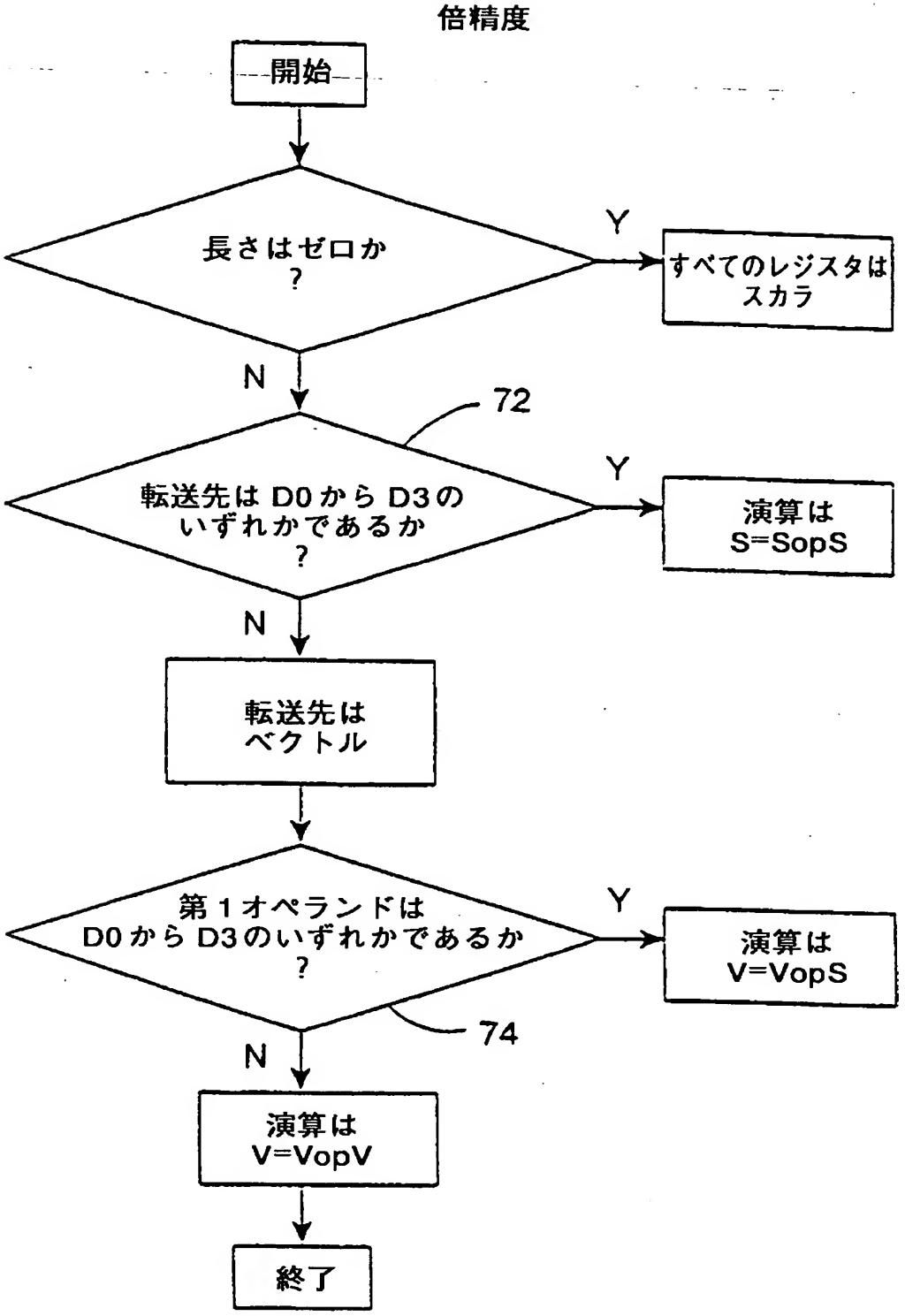
[Drawing 2]


加算 減算 乗算 除算
Inst0 Inst1 Inst2 Inst3
Inst4 Inst5 Inst6 Inst7
Inst8 Inst9 InstA InstB
S=S*S
V=V*S
V=V*V
演算コード
R1
R2
R3
44
MAC
40
ベクトルまたはスカラレジスタ
38
長さ/歩幅
46
52
3
3
3
ベクトル制御
50
54
レジスタ制御・命令発行
48
ロード/ストア制御
42
5
26

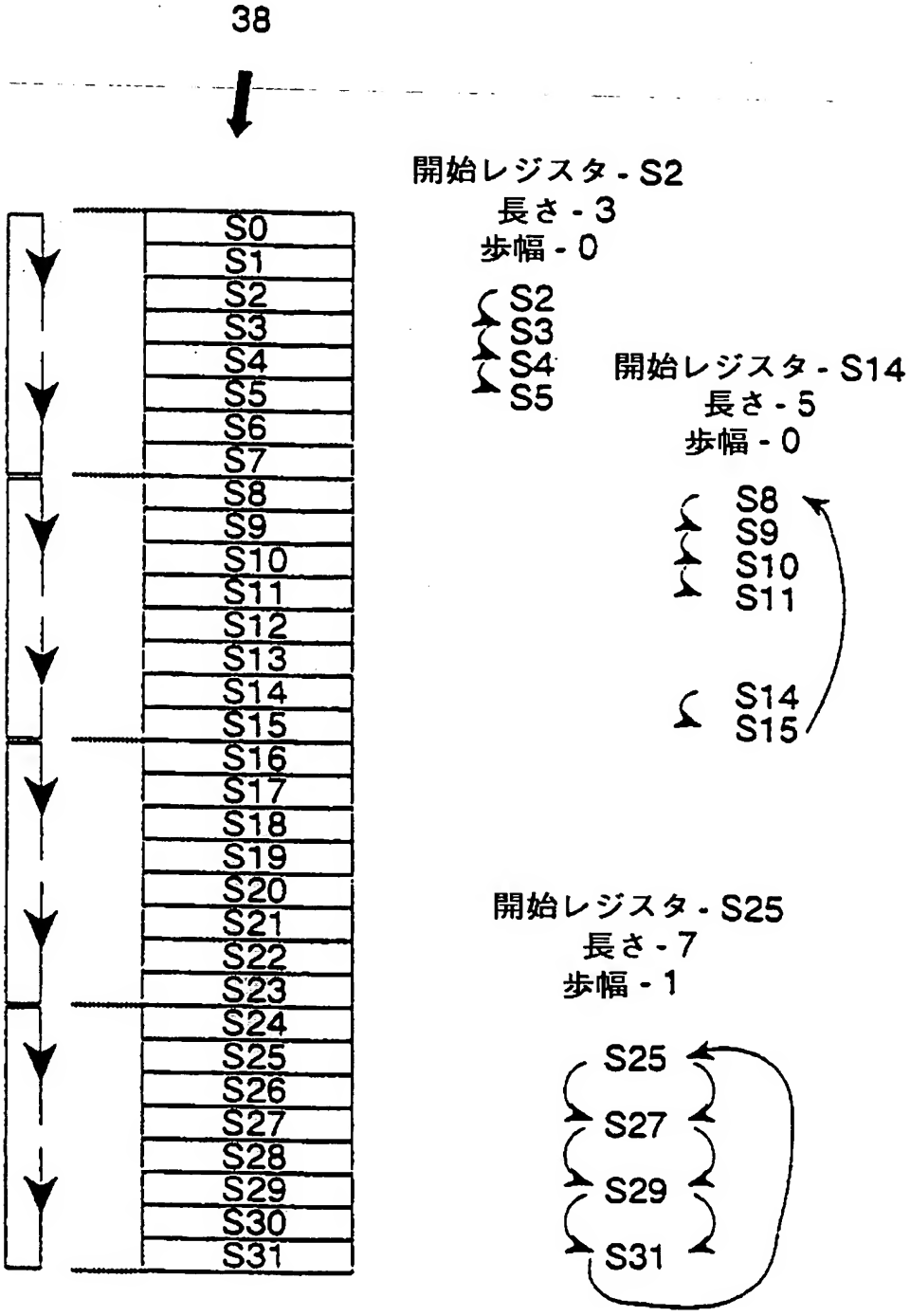
[Drawing 3]


単精度
開始
56
長さはゼロか
?
Y
58
すべてのレジスタ
はスカラ
N
60
転送先は S0 から S7 の
いずれかであるか
?
Y
62
演算は
S=SopS
N
転送先は
ベクトル
64
第１オペランドは
S0 から S7 のいずれかであるか
?
Y
68
演算は
V=VopS
66
N
演算は
V=VopV
70
終了

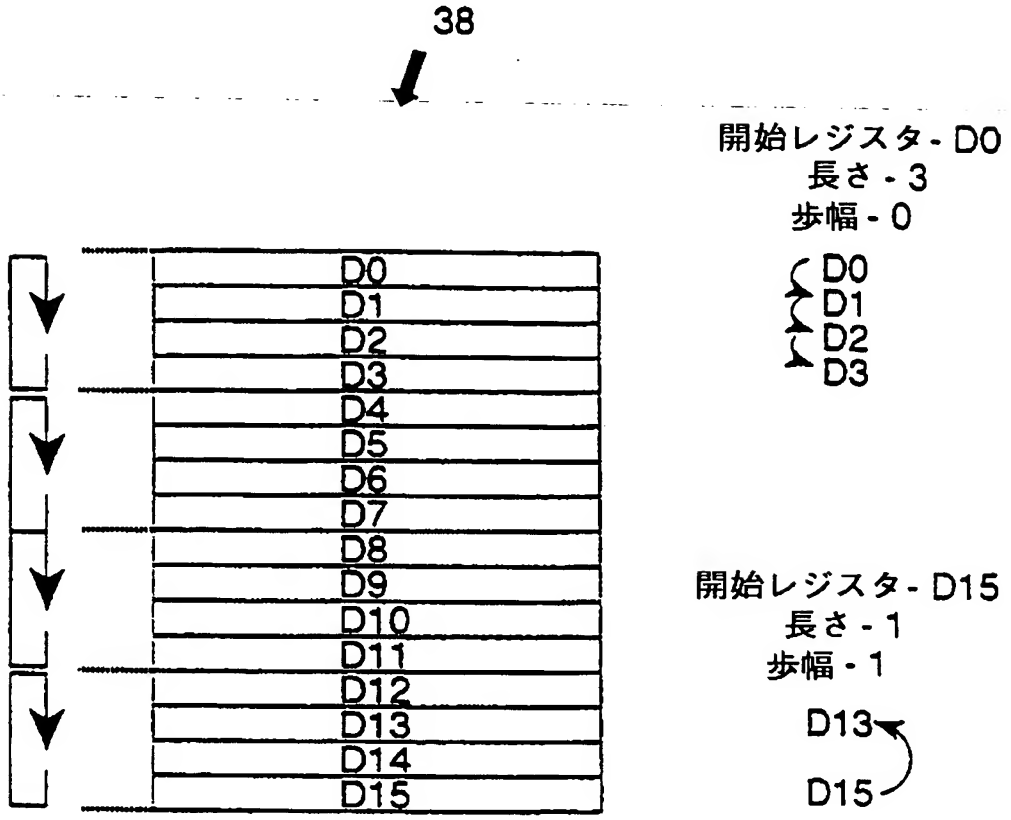
[Drawing 4]


倍精度
開始
長さはゼロか
?
Y
すべてのレジスタは
スカラ
N
72
転送先は D0 から D3 の
いずれかであるか
?
Y
演算は
S=SopS
N
転送先は
ベクトル
第 1 オペランドは
D0 から D3 のいずれかであるか
?
Y
演算は
V=VopS
74
N
演算は
V=VopV
終了

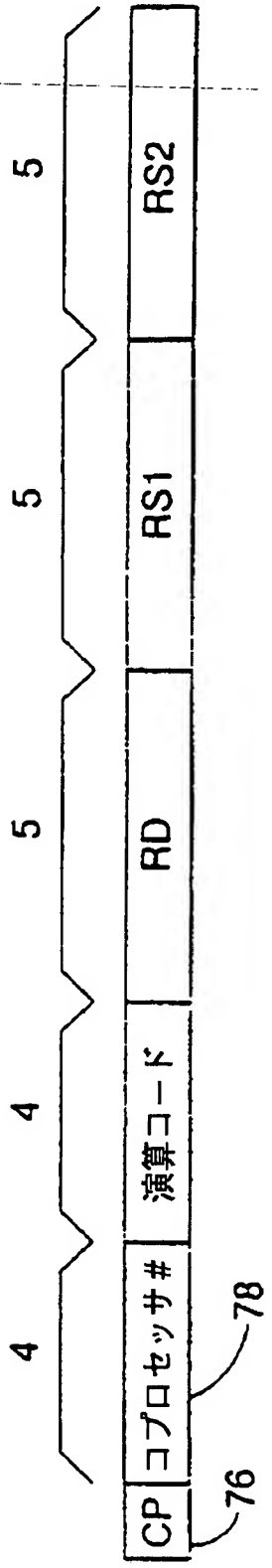
[Drawing 5]


38
開始レジスタ - S2
長さ - 3
歩幅 - 0
S2
S3
S4
S5
開始レジスタ - S14
長さ - 5
歩幅 - 0
S8
S9
S10
S11
S14
S15
開始レジスタ - S25
長さ - 7
歩幅 - 1
S25
S27
S29
S31
S0
S1
S2
S3
S4
S5
S6
S7
S8
S9
S10
S11
S12
S13
S14
S15
S16
S17
S18
S19
S20
S21
S22
S23
S24
S25
S26
S27
S28
S29
S30
S31

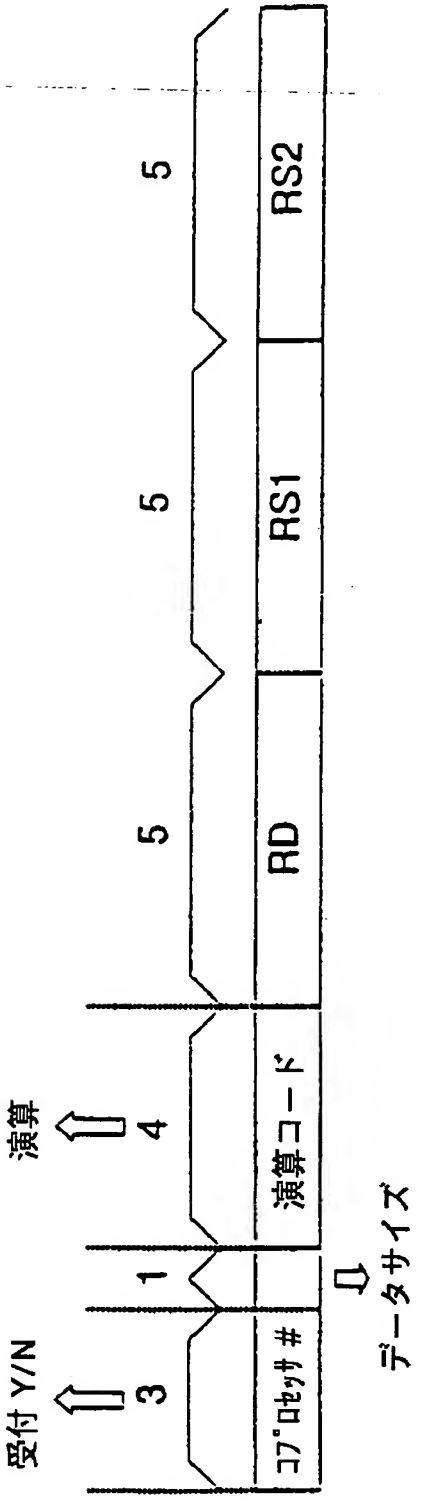
[Drawing 6]


38
D0
D1
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11
D12
D13
D14
D15
開始レジスタ - D0
長さ - 3
歩幅 - 0
D0
D1
D2
D3
開始レジスタ - D15
長さ - 1
歩幅 - 1
D13
D15

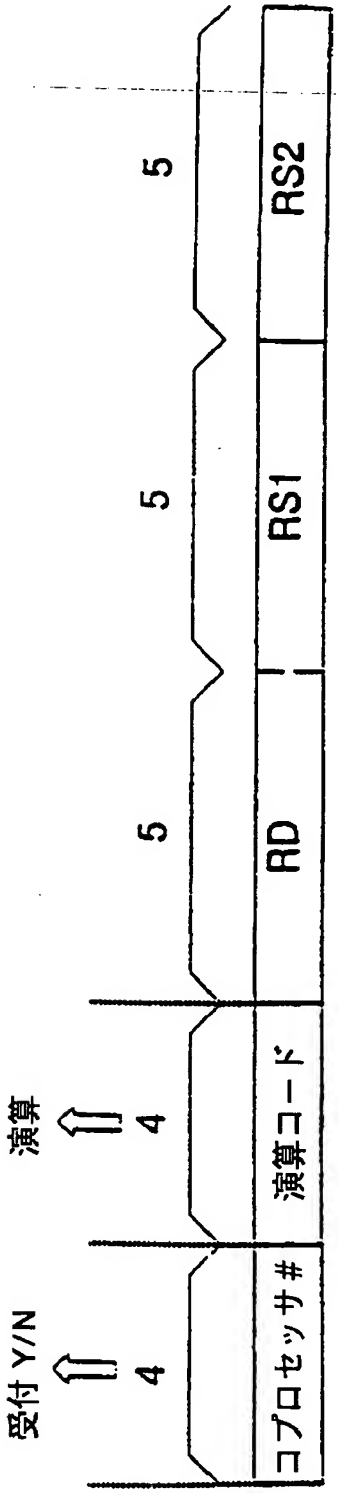
[Drawing 7 A]



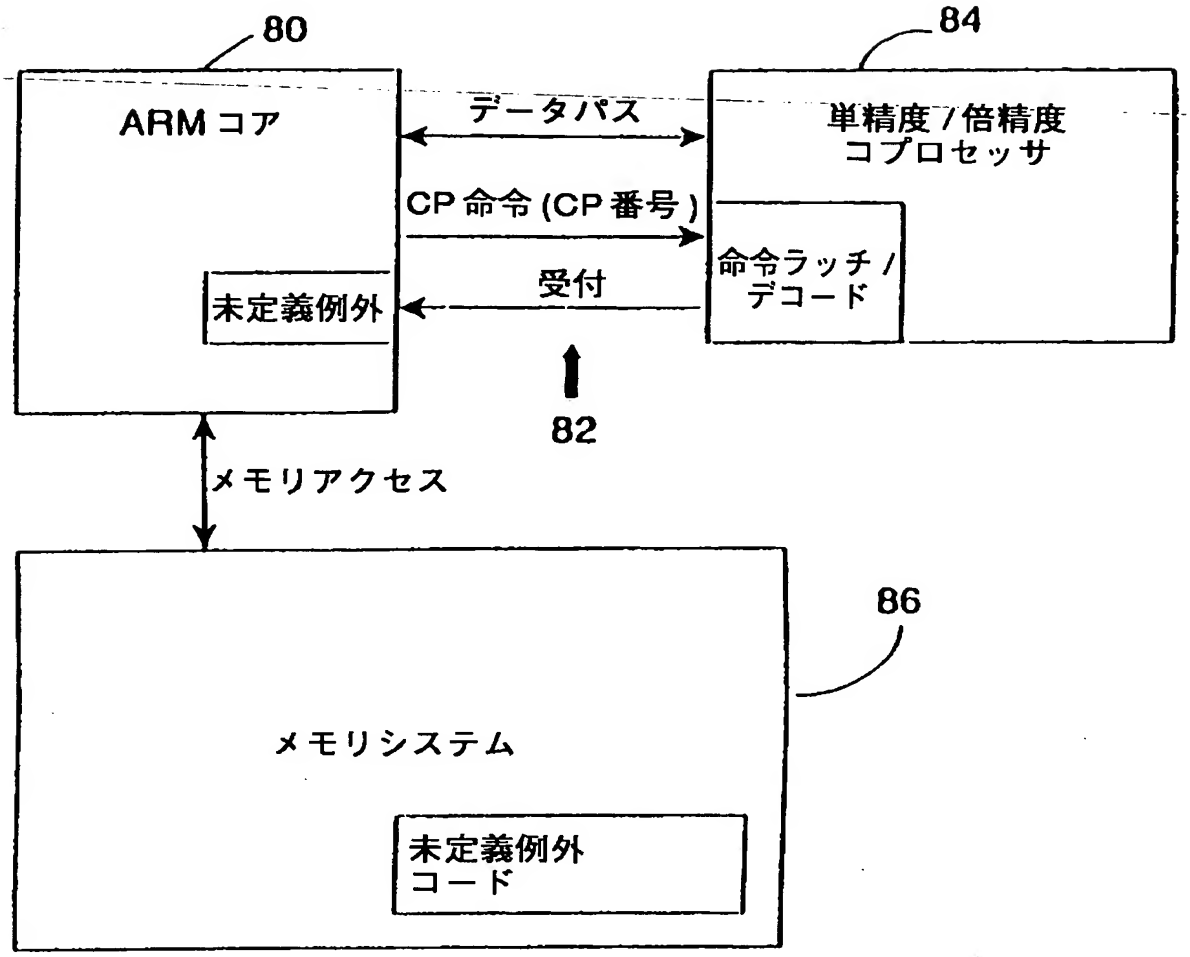
[Drawing 7 B]



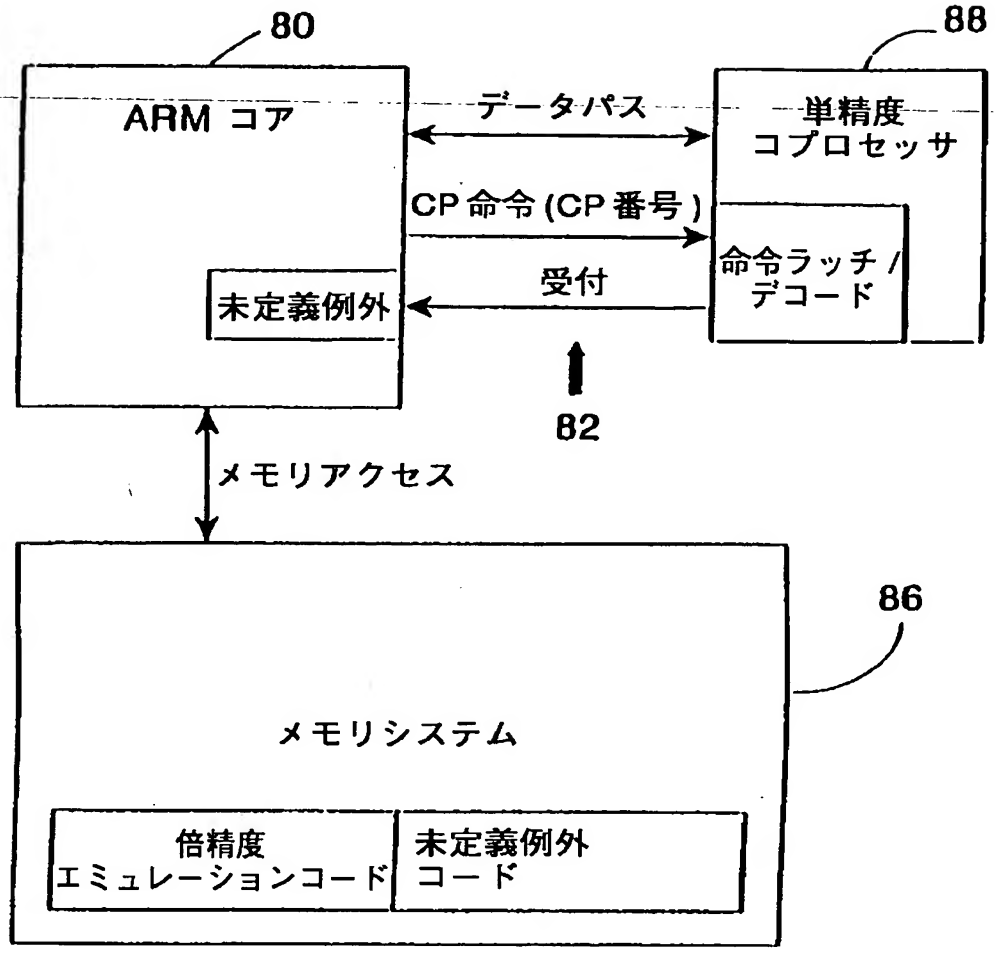
[Drawing 7 C]



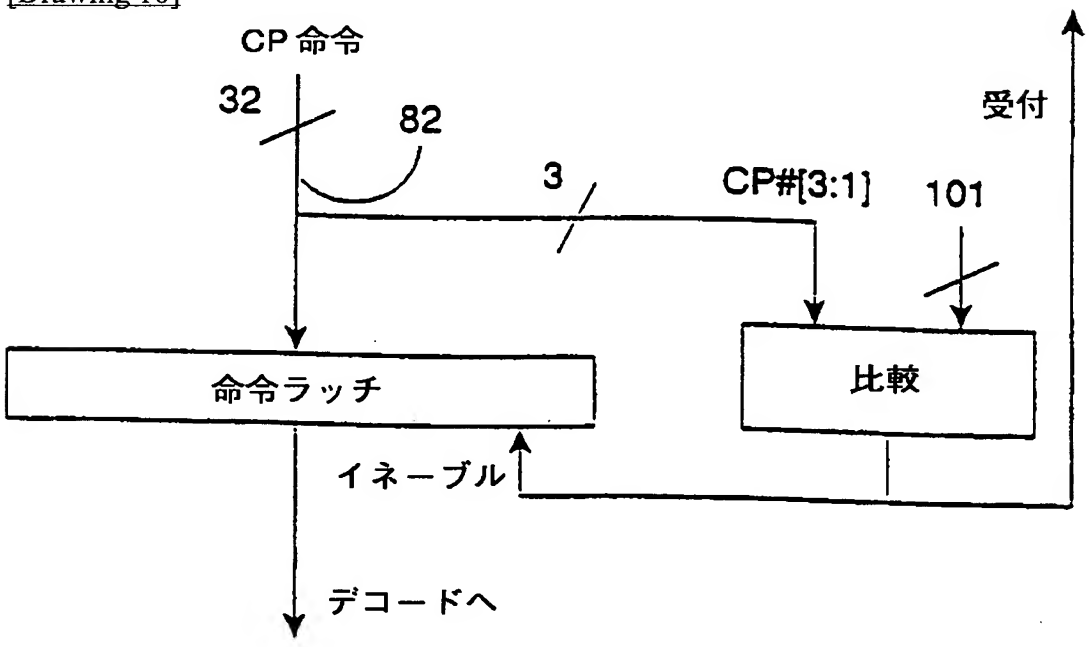
[Drawing 8]


80
ARM コア
データパス
84
単精度 / 倍精度
コプロセッサ
CP 命令 (CP 番号)
命令ラッチ /
デコード
受付
未定義例外
82
メモリアクセス
86
メモリシステム
未定義例外
コード

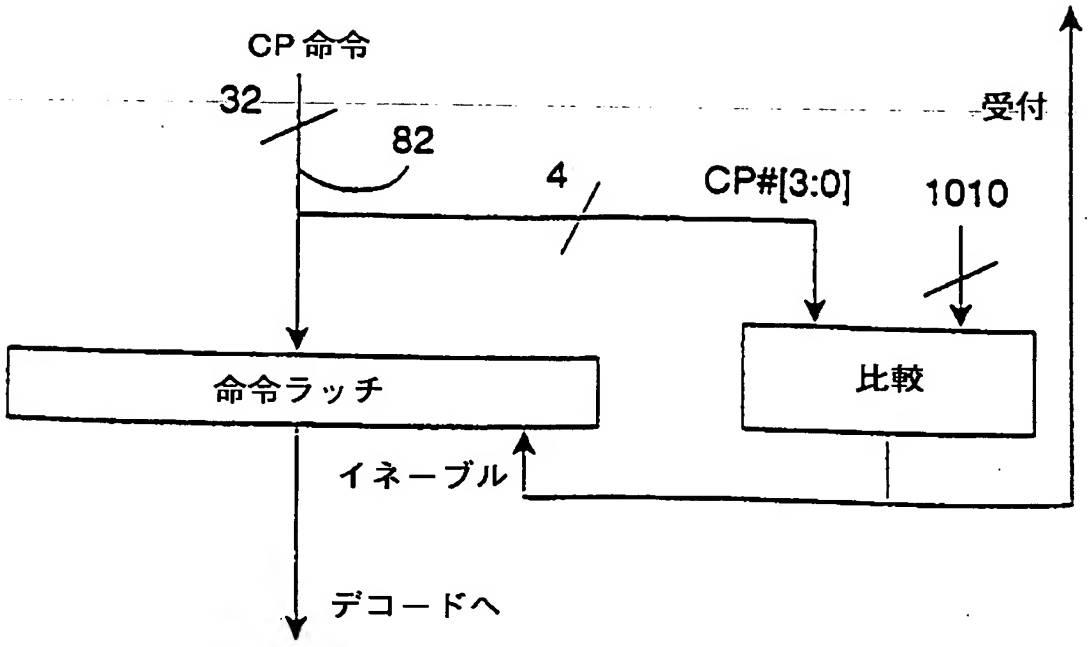
[Drawing 9]



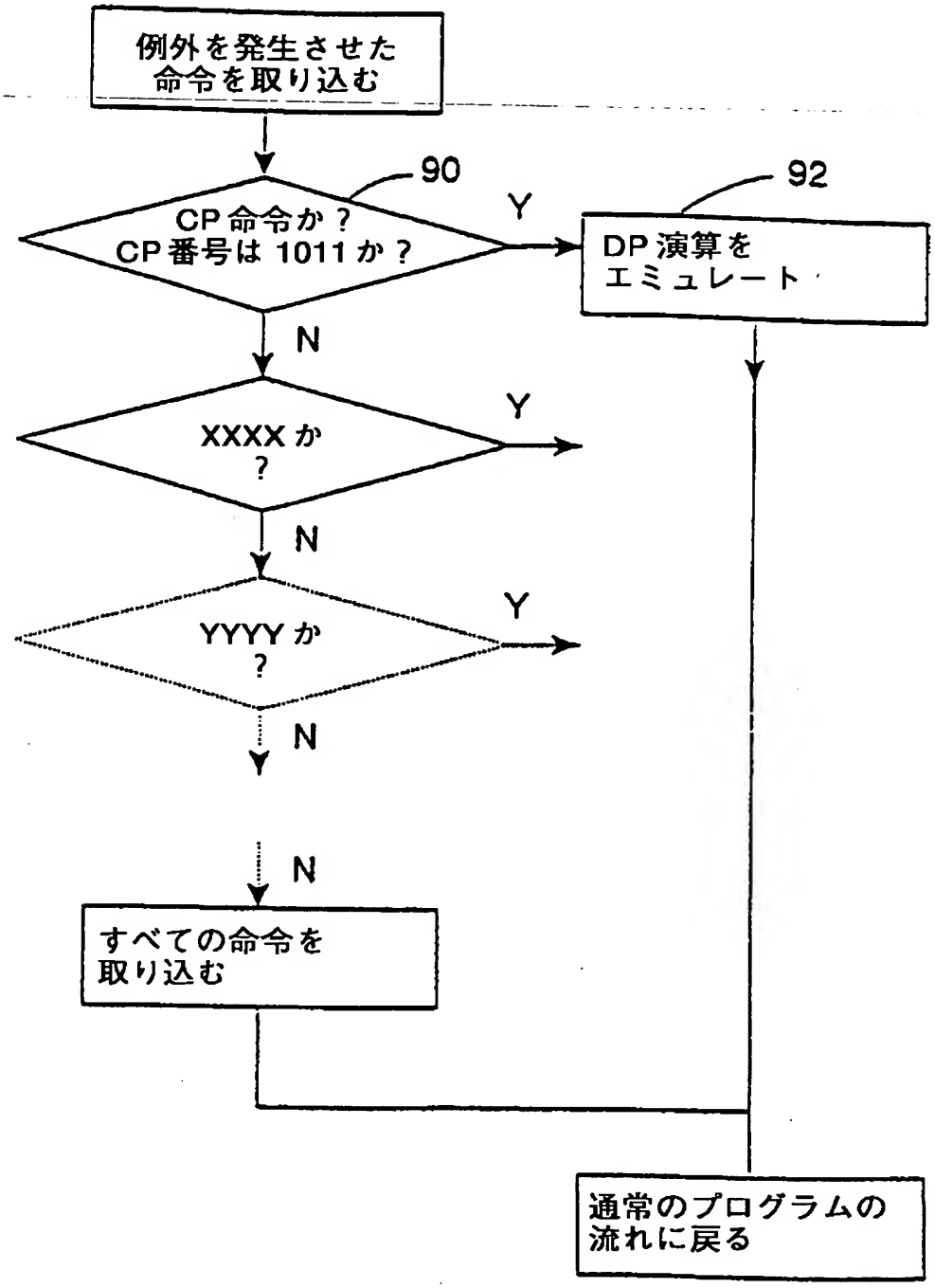
[Drawing 10]



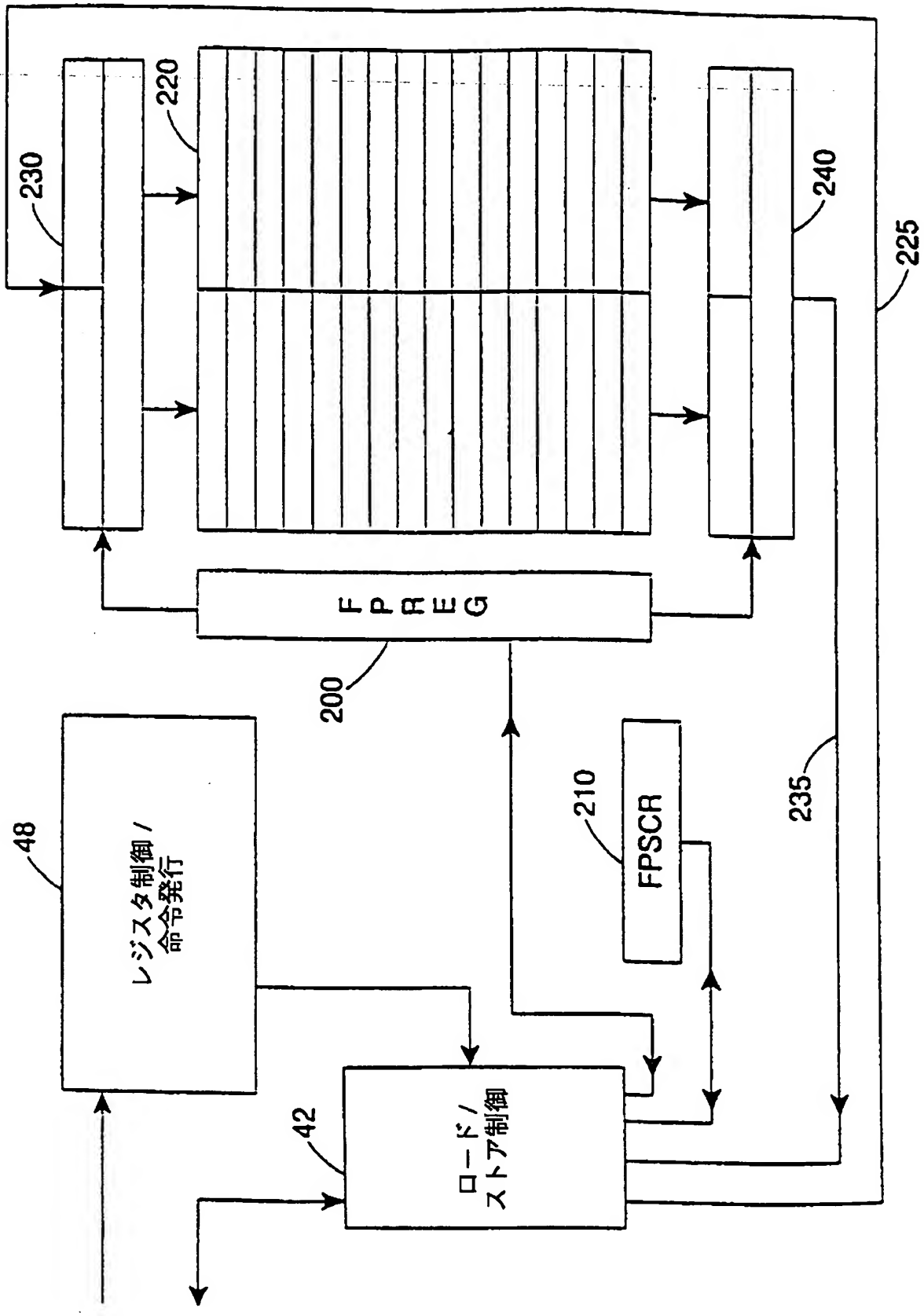
[Drawing 11]



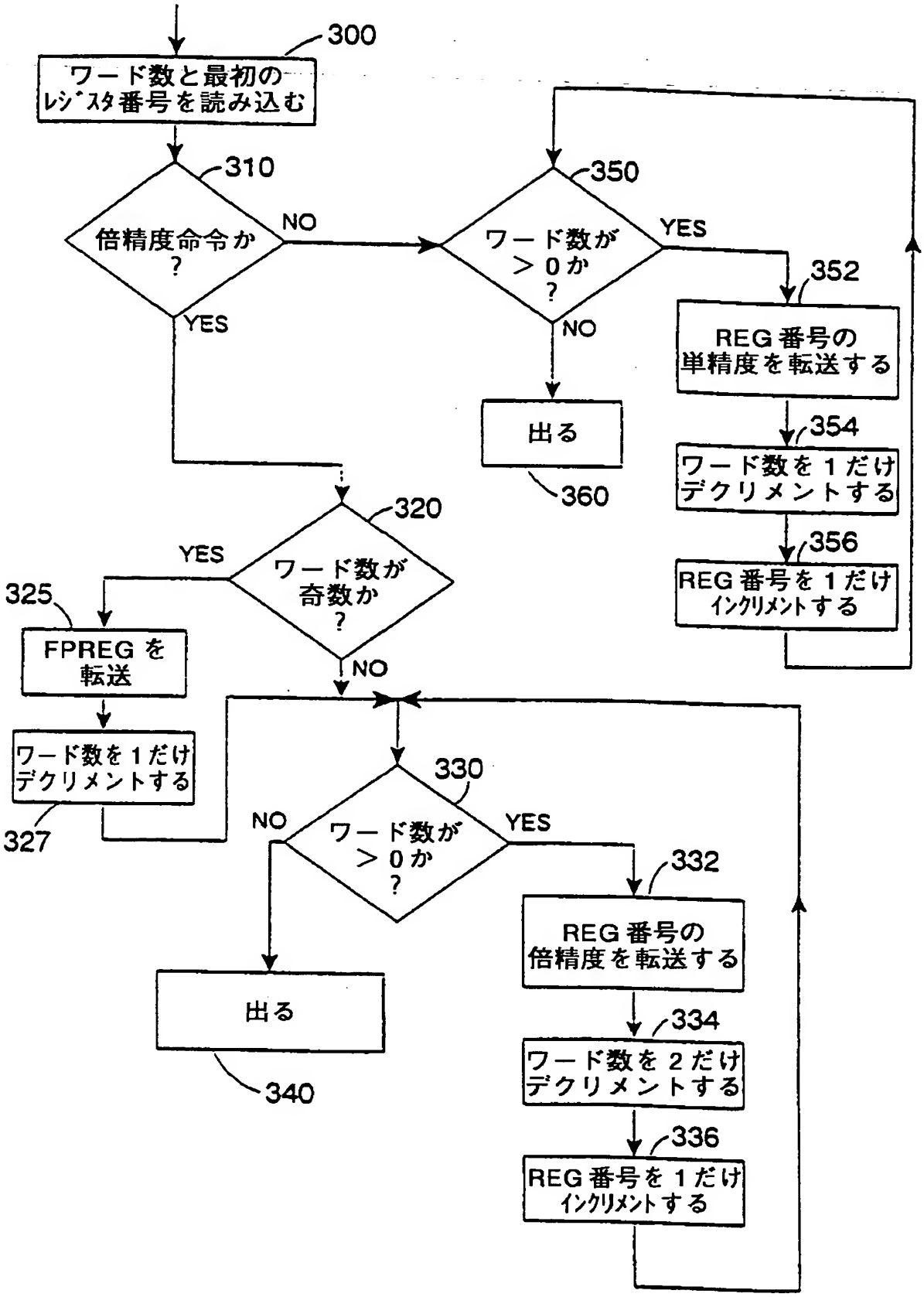
[Drawing 12]


例外を発生させた
命令を取り込む
90
CP命令か？
CP番号は1011か？
Y
92
DP演算を
エミュレート
N
XXXXか
？
Y
N
YYYYか
？
Y
N
N
すべての命令を
取り込む
通常のプログラムの
流れに戻る

[Drawing 13]


230
220
240
225
FPREG
200
210
FPSCR
235
48
レジスタ制御 / 命令発行
42
ロード / ストア制御

[Drawing 14]


300
ワード数と最初の
レジスタ番号を読み込む
310
倍精度命令か
?
NO
YES
350
ワード数が
＞0か
?
YES
NO
352
REG番号の
単精度を転送する
354
ワード数を1だけ
デクリメントする
356
REG番号を1だけ
インクリメントする
出る
360
320
ワード数が
奇数か
?
YES
NO
325
FPREGを
転送
ワード数を1だけ
デクリメントする
327
330
ワード数が
＞0か
?
NO
YES
332
REG番号の
倍精度を転送する
334
ワード数を2だけ
デクリメントする
336
REG番号を1だけ
インクリメントする
出る
340

[Drawing 15]

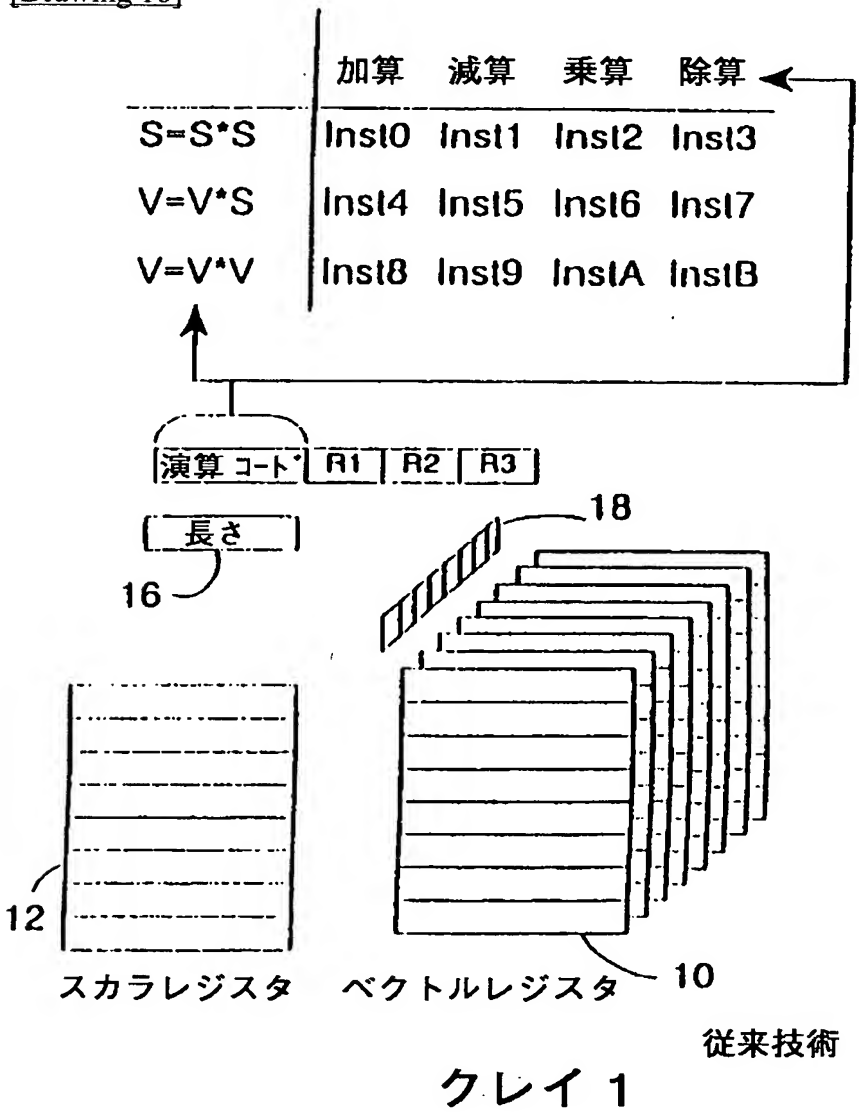
200

1
0
0
0
1
0

220

DH	DL
S	S
S	S
S	S
DH	DL
S	S

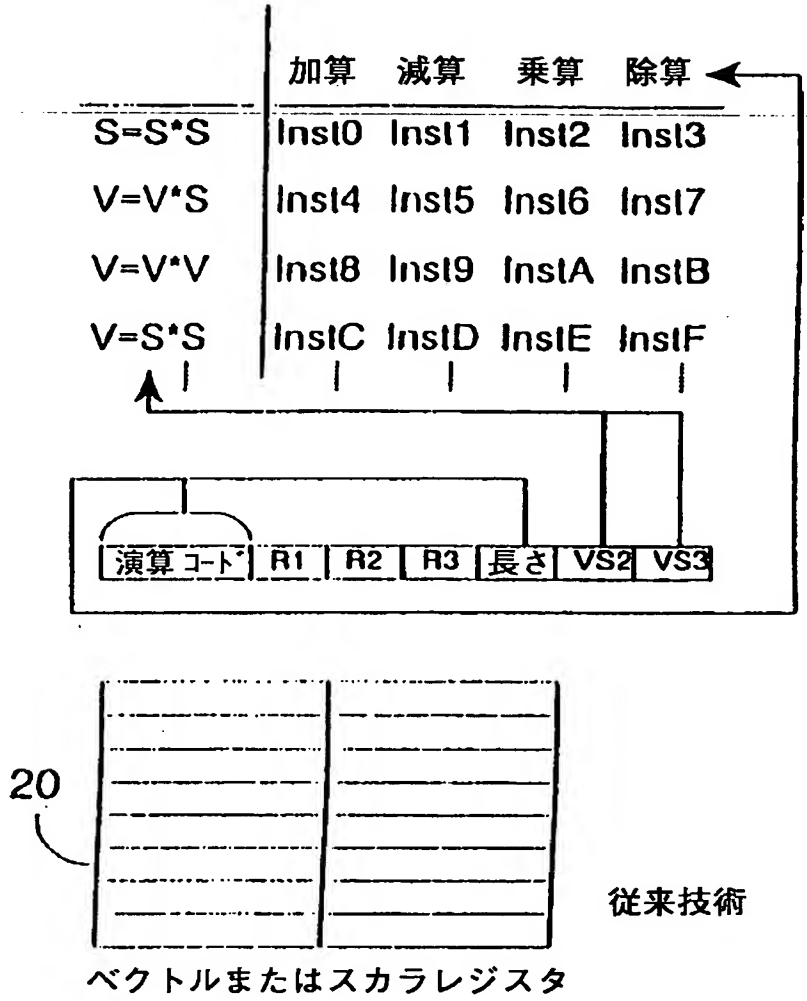
[Drawing 16]



従来技術

クレイ 1

[Drawing 17]



DEC マルチタイタン

[Translation done.]

* NOTICES *

1.This document has been translated by computer. So the translation may not reflect the original precisely.
2.**** shows the word which can not be translated.
3.In the drawings, any words are not translated.

WRITTEN AMENDMENT

[Procedure revision] The decodement presentation document of the 34th article amendment of Patent Cooperation Treaty
[Filing Date] March 9, Heisei 12 (2000. 3.9)
[Procedure amendment 1]
[Document to be Amended] Specification
[Item(s) to be Amended] Claim 1
[Method of Amendment] Modification
[Proposed Amendment]
[Claim 1] It is a data processor,
The register bank which has two or more accessible registers in the address,
It has an instruction decoder following at least one data-processing instruction which directs the vector operation which carries out multiple-times activation of the data-processing operation starting with the initial register directed to said data-processing instruction using the data value of a series of registers in said register bank,
As for said register bank, said a series of registers exist in said subset including at least one register subset,
Said instruction decoder is a data processor characterized by what said a series of registers are controlled for so that said a series of registers turn within said register subset.
[Procedure amendment 2]
[Document to be Amended] Specification
[Item(s) to be Amended] Claim 12
[Method of Amendment] Modification
[Proposed Amendment]
[Claim 12] The step which stores a data value in two or more accessible registers in the address of a register bank,
It is the data-processing approach of having the step which starts with the initial register directed to said data-processing instruction using the data value of a series of registers in said register bank following at least one data-processing instruction which directs vector operation, and carries out multiple-times activation of the data-processing operation,
As for said register bank, said a series of registers exist in said subset including at least one register subset,
It is the data-processing approach characterized by what said a series of registers turn around within said register subset during said activation.
[Procedure amendment 3]
[Document to be Amended] Specification
[Item(s) to be Amended] 0009
[Method of Amendment] Modification
[Proposed Amendment]
[0009]
The data processor [according to one viewpoint] by this invention,
The register bank which has two or more accessible registers in the address,
It has an instruction decoder following at least one data-processing instruction which directs the vector operation which carries out multiple-times activation of the data-processing operation starting with the initial register directed to said data-processing instruction using the data value of a series of registers in said register bank,
As for said register bank, said a series of registers exist in said subset including at least one register subset,

Said instruction decoder is characterized by what said a series of registers are controlled for so that said a series of registers turn within said register subset.
[Procedure amendment 4]
[Document to be Amended] Specification
[Item(s) to be Amended] 0010
[Method of Amendment] Modification
[Proposed Amendment]
[0010]
Since it was made to turn within the register subset of the number of registers (fewer than all the numbers of registers) of a register bank, the compact code which carries out the reuse of the data value in a register bank can be written without reloading or moving a data value. By performing a required surroundings lump by hardware, a data value can be processed in sequence which could start instruction code from the point of having differed in the subset whenever it used it, therefore is different, without using the excessive instruction for dividing a series of vector registers. Furthermore, data transfer to the data value of two or more registers which do not exist in a subset can be performed to coincidence by performing vector operation to the subset of a register around which it turns to these selves. A surroundings lump of a register is obtained also by offering the hardware which supports the ring (circulation) buffer mold configuration which incorporates data to a buffer and was read from the buffer again in two or more points which pursue the buffer of each other round and round, and suit.
[Procedure amendment 5]
[Document to be Amended] Specification
[Item(s) to be Amended] 0020
[Method of Amendment] Modification
[Proposed Amendment]
[0020]
The data-processing approach [according to another viewpoint] by this invention,
The step which stores a data value in two or more accessible registers in the address of a register bank,
It is the data-processing approach of having the step which starts with the initial register directed to said data-processing instruction using the data value of a series of registers in said register bank following at least one data-processing instruction which directs vector operation, and carries out multiple-times activation of the data-processing operation,
As for said register bank, said a series of registers exist in said subset including at least one register subset,
It is characterized by what said a series of registers turn around within said register subset during said activation.

[Translation done.]